VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2017 **Radim Kula**

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Simulace lokalizačního subsystému mobilního robotu

Localization subsystem simulation for mobile robot

2017 Radim Kula

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Zadání bakalářské práce

Student: **Radim Kula**

Studijní program: B2649 Elektrotechnika

Studijní obor: 2612R041 Řídicí a informační systémy

Téma: Simulace lokalizační subsystém mobilního robotu
Localization Subsystem Simulation for Mobile Robot

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Senzory pro lokalizaci mobilních robotů.
2. Existující lokalizační metody a algoritmy.
3. Prostředky pro tvorbu simulací (C++, C#, Matlab).
4. Návrh vlastního řešení.
5. Implementace vybraného lokalizačního algoritmu.
6. Testování.
7. Zhodnocení dosažených výsledků závěrečné práce.

Seznam doporučené odborné literatury:

[1] KONEČNÝ, Jaromír. *Principy řízení mobilních servisních robotů*. Ostrava, 2014. Disertační práce. VŠB-Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, Katedra kybernetiky a biomedicínského inženýrství.
[2] KURECKA, A., J. KONECNY, M. PRAUZEK a J. KOZIOREK. Monte carlo based wireless node localization. *Elektronika ir Elektrotechnika,* 2014, vol. 20, no. 6, pp. 12-16. DOI: http://dx.doi.org/10.5755/j01.eee.20.6.7260. ISSN 1392-1215. Dostupné z: http://www.eejournal.ktu.lt/index.php/elt/article/viewFile/7260/3678.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jaromír Konečný, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Ing. Jiří Koziorek, Ph.D.
vedoucí katedry

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * 1D16 * OSTRAVA *

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. dubna 2017

.....

Rád bych na prvním místě poděkoval vedoucímu mé bakalářské práce, panu Ing. Jaromíru Konečnému, Ph.D. za obrovskou trpělivost, rady a připomínky při tvorbě bakalářské práce.

Také bych rád poděkoval svým kamarádům, rodině za podporu nejen při tvoření bakalářské práce, ale i při průběhu celého studia.

Abstrakt

Tato práce řeší implementaci lokalizačního algoritmu pro lokalizační subsystém mobilního robotu. Výstupem práce je simulační program v jazyce C#, který simuluje chování senzorů, pohyb a lokalizační algoritmus založený na pravděpodobnosti. Konkrétně se jedná o algoritmus Monte Carlo. Simulační software obsahuje možnost výběru mapy, grafické zobrazení průběhu simulace, krokování lokalizačního algoritmu Monte Carlo, LiDAR pro měření vzdálenosti, nastavení šumu a nastavení hustoty částic pro MCL.

Klíčová slova: lokalizace Monte Carlo, pravděpodobnostní lokalizace, částicový filtr, LiDAR, mobilní robot.

Abstract

The thesis deals with an implementation of a localization algorithm for a localization subsystem of a mobile robot. The outcome is a simulation program in C# language which simulates sensor behaviour, motion and localization algorithm based on probability. Specifically, it is Monte Carlo algorithm. The simulation software contains a possibility to choose a map, graphic projection of a simulation development, debuging of the Monte Carlo localization algorithm, LiDAR for distance measuring, noise level setting and density of particles for MCL setting.

Key Words: Monte Carlo localization, probability localization, partice filter, LiDAR, mobile robot.

Obsah

Seznam použitých zkratek a symbolů

B	–	Magnetická indukce
C++	–	Vyšší programovací jazyk
CCD	–	Charge-coupled device – elektronická součástka používaná pro snímání obrazové informace
C#	–	Objektově orientovaný jazyk
DC	–	Direct Current – stejnosměrný proud
DPS	–	Digitální signálový procesor
FOV	–	Field of view – Zorné pole
GPS	–	Global Positioning Systém – družicový systém pro určení polohy
IMU	–	Inertial measurement unit – Inerciální měřici jednotka
LiDAR	–	Light Detection And Ranging - Laserový měřič vzdálenosti
MC	–	Metoda Monte Carlo
MCL	–	Monte-Carlo Localization – Monte Carlo lokalizace
MHz, GHz	–	Jednotka frekvence — megahertz, gigahertz
POS	–	Positioning and orientation systems – polohovací systém
RPM	–	Otáčky za minutu
I	–	Stejnosměrný proud
S	–	Směrodatná odchylka
$S_{\text{čk}}$	–	Obsah čtvrtiny kruhu
$S_{\text{č}}$	–	Obsah čtverce
T	–	Těžiště
T_{e}	–	koeficient chyby posuvu
U_{h}	–	Hallovo napětí
V	–	Volt – Jednotka napětí
W_{B}	–	Nenormalizovaná váha částice
W_{max}	–	Maximální odchylka od skenu
W_{10}	–	10% W_{max}
W_{30}	–	30% W_{max}
X_i,X_{T}	–	Moduly pro vypočet směrodatné odchylky
d	–	Délka posuvu robotu
d_{Ai}	–	Délka paprsku lidaru na hlavním robotu
d_{Bi}	–	Délka paprsku lidaru na pravděpodobném robotu
d_{e}	–	Vzdálenost težiště od hlavního robotu
m	–	Množina částic
$n_{\text{čk}}$	–	Počet částic uvnitř čtvrtiny kruhu
$n_{\text{č}}$	–	Počet částic uvnitř čtverce

l,l_i	–	Poloha částice
r	–	Délka strany čtverce, poloměr kruhu
w_i	–	Váha částice MCL
w_s	–	Šířka stěny
x,y	–	Obecné parametry, pozice robotu
x_As,y_As	–	Počáteční souřadnice stěny
x_Bs,y_Bs	–	Konečné souřadnice stěny
x_A,y_A	–	Počáteční souřadnice robotu
x_B,y_B	–	Konečné souřadnice robotu
x_i,y_i	–	Souřadnice pravděpodobné pozice částice m_i
x_T,y_T	–	Souřadnice těžiště
x_R,y_R	–	Souřadnice hlavního robotu
δ	–	Diskrétní Diracův impulz
θ	–	Úhel natočení robotu
θ_A	–	Počáteční úhel natočení robotu
θ_B	–	Konečný úhel natočení robotu
i	–	Proměnná cyklu
k	–	Index posloupnosti

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V současné době se s rozvojem odvětví robotiky začíná čím dál více vyskytovat potřeba autonomního řízení. Příkladem jsou třeba osobní auta, která se dokážou řídit bez pomoci řidiče nebo také robotické sekačky. Základním problémem tohoto řízení je schopnost robotu se lokalizovat v prostoru a najít optimální trasu své cesty. Tento problém vyžaduje určitou výpočetní náročnost, kterou řeší rozvoj počítačových procesorů, mikroprocesoru, senzorů atd. Cílem této bakalářské práce je pochopení principu lokalizace mobilních robotů ve známém prostředí. Tato práce popisuje různé lokalizační postupy, zejména se zaměřuje na metodu Monte Carlo. Tuto část řeší kapitola 3.

Dalším cílem je seznámení s potřebnými senzory, které jsou pro lokalizaci nezbytné, jako jsou například měřiče vzdálenosti robotu od překážky nebo stěny. Ty mohou být například ultrazvukové nebo laserové. Dále se bude kapitola 2 zabývat senzory pro zjištění pootočení kol neboli enkodéry.

Kromě senzoru práce řeší v kapitole 4 i různé programovací nástroje a jazyky, ve kterých můžeme chování robotu simulovat. Konkrétně se bude zabývat programovacími jazyky C#, Matlab a C++.

Hlavním cílem je vytvoření simulačního programu, který bude simulovat chování robotu osazeného několika senzory pro možnost implementace a testování vybraného lokalizačního algoritmu. Tato problematika je rozdělena do tří kapitol, kdy kapitola 5 bude řešit simulaci senzoru robotu, ať už pro posuv nebo měření vzdálenosti, kapitola 6 se bude zabývat implementaci MCL a kapitola 7 popisuje testování chování simulace za různých podmínek, jako jsou nastavení přesnosti senzoru nebo samotného lokalizačního algoritmu.

Posledním cílem této práce je zhodnocení vlivu přesnosti senzoru a nastavení MCL na průběh rychlosti robotu se lokalizovat v prostoru-kapitola 8.

2 Senzory pro lokalizaci mobilních robotů

K tomu, abychom mohli roboty lokalizovat, musí mít robot nějaké zdroje informací o ujeté vzdálenosti pootočení nebo vzdálenosti od překážky, zdi a podobně. Pro tento účel slouží senzory, jako jsou:

- Enkodéry
- Hallův senzor
- Ultrazvukové senzory
- Laserové dálkoměry

2.1 Enkodéry

Enkodéry jsou převodníky mechanického pohybu na elektricky signál, jehož výstup odpovídá změřené hodnotě kódovaného vzoru na rotačním disku nebo pohyblivé šabloně. Enkodéry jsou klasifikovány podle [4]:

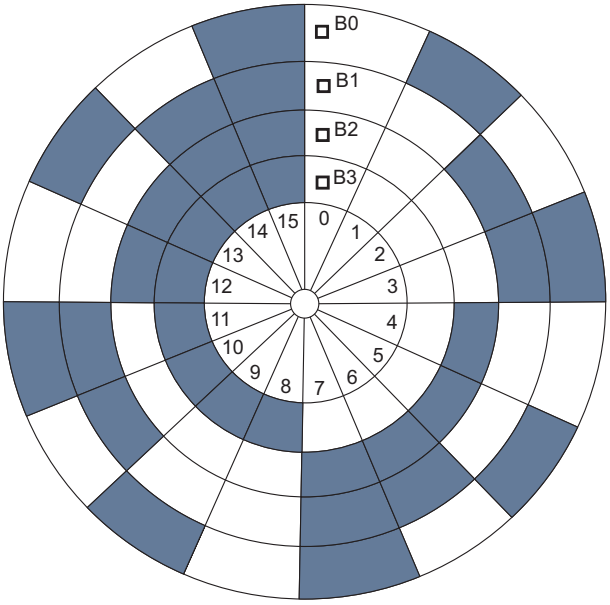
- metody použití na čtení kódového elementu: kontaktní, bezkontaktní
- typu výstupu: absolutní poloha nebo inkrementační pulzy
- fyzikálního základu použitého pro snímaní: vodivost, magnetismus, optika nebo kapacita

2.1.1 Kontaktní enkodéry

Kontaktní enkodéry jsou takové, které využívají mechanický kontakt mezi kartáčem nebo pinem senzoru a kódovaným diskem. Disk obsahuje sérii koncentrických kruhů nebo drah, což jsou tenké metalické proužky-jejich pospojování ukazuje obrázek 1. Čtyři dráhy ukázané na obrázku 1 reprezentují binární kód skládající se z $2^0, 2^1, 2^2, 2^3$. K nim přidružené senzory jsou označené jako B_0, B_1, B_2, B_3 a enkódují čísla od 0 do 15. Jak se disk otáčí, senzory střídavě spínají vodivé proužky a přilehlé izolátory, tím vytvářejí sérii obdélníkových průběhů.

Uniformované a neuniformované vzory disku jsou použité v závislosti na aplikaci. Virtuálně každý vzor, který je možné vytvořit fotograficky, může být použitý na disk enkodéru. Typická aplikace je měření pozice hradel, které vyžaduje uniformovaný vzor. Každá neuniformovanost disku je zdrojem chyb. Neuniformovaný odstup segmentu produkuje chyby polohy, excentricita způsobuje chybu, která je sinusoidní funkcí úhlu natočení hradel.

Výkonové specifikace jsou limitované vzhledem na faktory, jako je praktické rozložení segmentu na disk, přemostění segmentů a opotřebení kontaktů [4].



Obrázek 1: Enkodérový disk absolutní polohy.

2.1.2 Bezkontaktní enkodéry

Bezkontaktní enkodéry jsou ty, které využívají jiné fyzikální jevy než elekktrickou vodivost- například magnetické, kapacitní nebo optické.

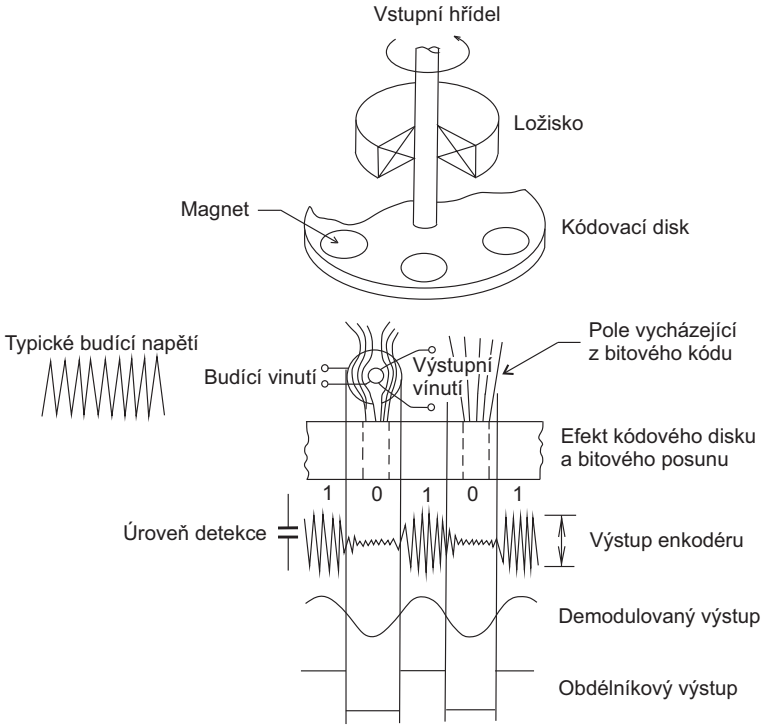
2.1.3 Magnetické enkodéry

Magnetické enkodéry byly vyvinuty jako náhrada místo kontaktních enkodérů v aplikacích limitovaných rychlostí otáčení. Magnetické enkodéry fungují na principu dekódování změny rezonanční frekvence, změny magnetizace nebo magnetické saturace induktoru. Pro každou metodu indukční tok magneticky kódovaného disku ovlivňuje změnu napomáháním nebo tlumením současného stavu. To znamená, že pro každý princip existují dva stavy, které odpovídají logické jedničce a nule.

Typ založený na rezonanční frekvenci využívá laděného obvodu, jehož frekvence představuje jeden logický stav, a rozladění obvodu představuje opačný logický stav.

V metodě využívající magnetickou saturaci je induktor buď saturovaný nebo ne. Střídavě magnetický odpor obvodu je efektivně přeložený na logické jedničky a nuly.

Rozlišení je limitované velikostí zmagnetizovaného místa a komplikovanou interakcí mezi zmagnetizovanými přilehlými drahami na disku. Magnetické enkodéry překonaly nízkou rychlost limitující kontaktni enkodéry a nabídly vyšší životnost pomocí eliminování fyzického kontaktu mezi diskem a senzorem. Taktéž magnetické enkodéry fungují spolehlivě v náročnějších podmínkách snímání. A však vysoké okolní magnetické toky nebo hustota vyzařování mohou poškodit vzor disku nebo vyvolat stav saturovaného jádra. Vyšší opatření proti elektromagnetické inter-



Obrázek 2: Typické magnetické kódovaní.

ferenci jsou vyžadována, když jsou magnetické enkodéry použity v systému. Obrázek 2 ilustruje princip typického stavového magnetického enkodování [4].

2.1.4 Kapacitní enkodéry

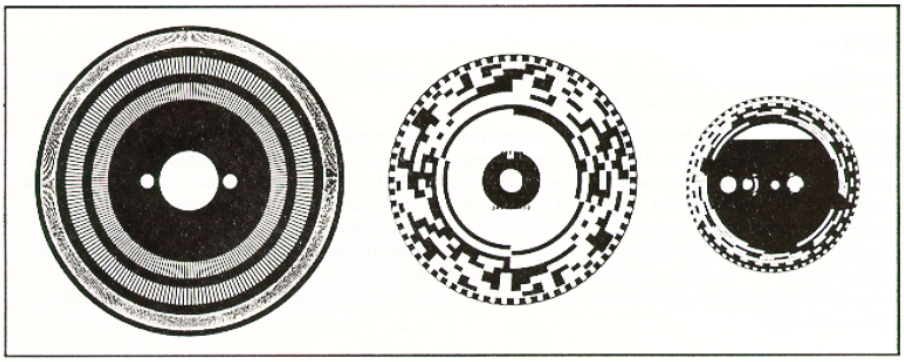
Kapacitní enkodéry jsou nejméně používany z bezkontaktních typů a byly vyvinuty jako reakce na speciální požadavky. Odečet se provádí elektrostaticky pomocí fázového posunu nebo kontrolou frekvence k získání digitálního výstupu.

Protože kapacitní zařízení nejsou běžně dostupná jako standardní vybavení, byly vyrobeny jednootáčkové enkodéry. Teoreticky, kapacitním způsobem je možné uskutečnit každou úlohu, ve které se používají kontaktní kapacitní nebo optické enkodéry. A však praktické problémy návrhu výroby a prodeje omezily používání kapacitního snímání [4].

2.1.5 Optické enkodéry

Optický enkodér byl první z nekontaktních zařízení na odstranění problémů s opotřebením souvisejícím s kontaktními enkodéry. Dnešní optické enkodéry poskytují nejvyšší rozlišení a přesnost a mohou být efektivně používané ve vysokých rychlostech.

Optické enkodéry mají průsvitné a absorpční segmenty (obrázek 3). Disky mohou být vyrobeny expozicí fotografické emulze světla, vrstvením kovu na substrátě nebo frézováním elementu



Obrázek 3: Disky enkodéru [4].

do kovového substrátu. Každý typ má svoje charakteristiky, které ho dělají vhodným pro určitý typ aplikace.

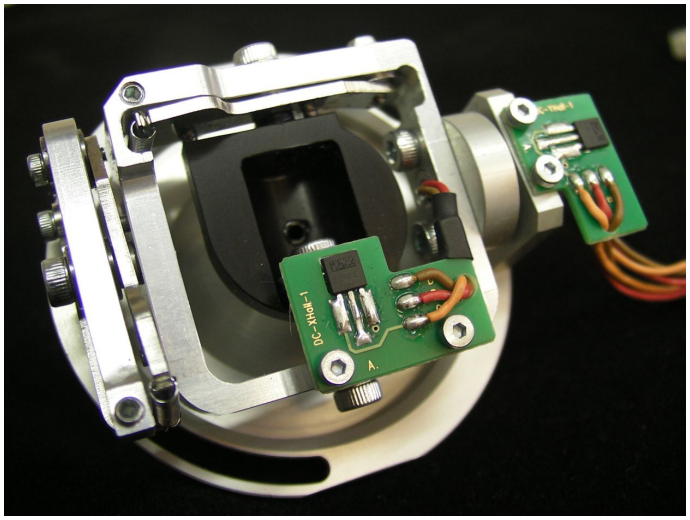
Čtení lze uskutečnit polem přesně seřazených optických senzorů uložených na jedné straně disku. Zdroj světla na opačné straně poskytuje osvětlení pro senzory. Jak se disk otáčí, v odezvě na vstupní proměnné neprůhledna místa na disku přecházejí mezi světly a modulují výstup senzoru s ohledem na zvolené kódování kotouče. Optické systémy zaostřují světlo na senzory. Led zdroje světla, zrcadla, křišťály, čočky, opticka vlákna, laserové diody a optické štěrbiny nebo difrakční mřížky splňují tuto funkci. Detekce světla mohou být uskutečněné jedním z více zařízení. Materiály všech typů detekčních zařízení jsou vybrané ze skupin III, IV, V periodické tabulky a leží polovinou spektra mezi kovem a nekovem.

2.2 Hallův senzor

Patří mezi magnetogalvanické snímače a využívá tzv. Hallova jevu v polovodičích a vodivých materiálech. Tento snímač je ovlivňován působením Lorentzovy síly. Hallův jev je založen na působení magnetického pole na polovodivou desku, která vlivem tohoto pole generuje Hallovo napětí U_h. Toto napětí jde spočítat jednoduchým vzorcem (1), kde k je konstanta, která zahrnuje materiál a tloušťku desky, I je stejnosměrný proud a B magnetická indukce způsobena magnetickým polem.

$$U_h = k \cdot I \cdot B \tag{1}$$

Tento senzor se používá k zjištění otáček motoru, kola nebo také k zjištění polohy hřídele. Na kole mohou být upevněny magnety, které při otáčení ovlivňují Hallův senzor, pomocí kterého poté mikroprocesor vyhodnotí „peak“ při otočení kola a následně i otáčky. Pro použití jako senzor polohy ho hojně využívá firma JETI model při výrobě svých RC vysílačů (viz obrázek 4). Výhodou tohoto senzoru je bezkontaktní měření polohy, což značně snižuje jeho poruchovost.

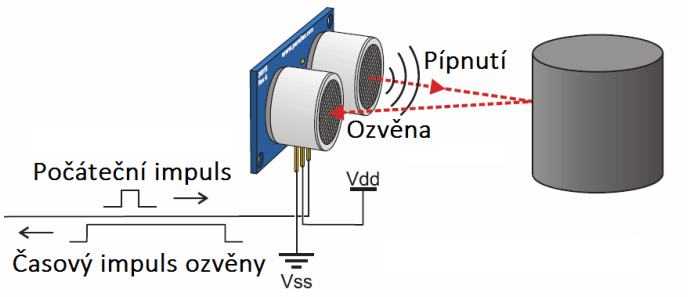


Obrázek 4: Příklad použití Hallovy sondy [13].

2.3 Ultrazvukové senzory

Princip tohoto senzoru (viz obrázek 5) spočívá ve vyslání impulzu o vysoké frekvenci[1], který se šíří prostorem rychlostí zvuku[2] a při nárazu na stěnu nebo překážku se vrací zpátky k čidlu jako ozvěna. Řídící systém ultrazvuku spočítá čas mezi vysláním impulzu a přijatým signálem, a ten přepočítá na vzdálenost. U tohoto senzoru se musí zahrnout prostředí, ve kterém vzdálenost měří, a podle toho počítat vzdálenost [17].

Využívá se v prašném prostředí, kde by optické senzory vzdálenosti nebyly přesné. V mobilní robotice jsou ultrazvukové senzory velmi oblíbené, neboť jejich cena je velice nízká a je jednoduché je používat. Mezi nevýhody patří nižší rychlost než je u optických senzorů. Při použití více ultrazvukových senzorů hrozí rušení a zkreslení výsledků [17].



Obrázek 5: Princip Ultrazvukového senzoru [18].

Jako příklad ultrazvukového senzoru je uveden velice rozšířený HC–SR04 od firmy ELEC Freaks, viz obrázek 6. Tento produkt je napájený 5 V DC, měří do maximální vzdálenosti čtyř

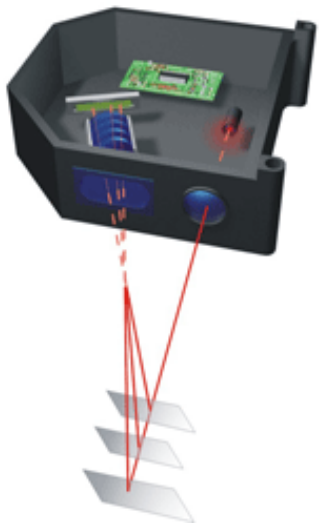
[1] Vysoká frekvence, která přesahuje 20 kHz.
[2] V klasickém prostředí je rychlost zvuku $340\,\mathrm{m}\cdot\mathrm{s}^{-1}$.

Obrázek 6: Ultrazvukový senzor HC-SR04.

metrů. Pro svou funkci využívá impulz od délce $10\,\mu$s při rychlosti $40\,rmkHz$. Uhel rozsahu měření je 15° [16].

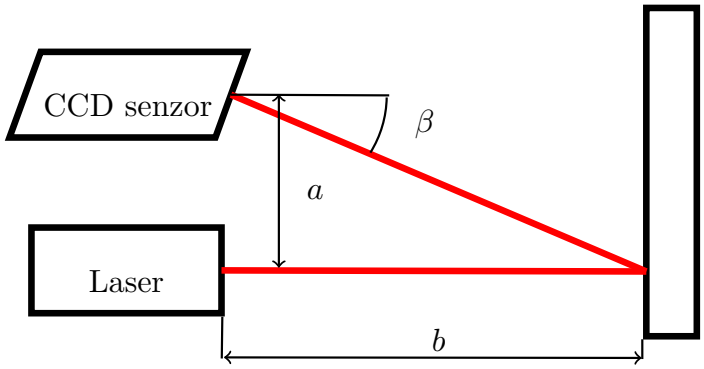
2.4 Laserové triangulační dálkoměry

Triangulační princip patří mezi moderní metody měření vzdálenosti. Tato metoda využívá odrazu paprsku od překážky při konstantním úhlu, přičemž vzdálenost dopadu odraženého paprsku určuje vzdálenost senzoru od překážky, a to daleko přesněji, než při měření času letu světla nebo jeho intenzity. Přesnost této metody závisí pouze na kvalitě vyhodnocovacího senzoru, který měří uhel dopadaného paprsku-příklad takového senzoru můžeme vidět na obrázku 7.



Obrázek 7: Triangulační laserový senzor [12].

Paprsek vychází většinou z polovodičové laserové diody, která má dostatečnou intenzitu. Tento paprsek dopadne na překážku, kde vytvoří bod světla. Z bodu světla se zpětně odrazí paprsek, který dopadá na vstupní čočku optického přijímače pod určitým úhlem. Průchod paprsku čočkou vyhodnocuje CCD senzor. Princip vyhodnocování vzdálenosti můžeme vidět na obrázku 8, kde k dopočítání vzdálenosti od překážky slouží vzorec (2).



Obrázek 8: Princip triangulační metody.

$$\text{cotan}(\beta) \cdot a = b, \tag{2}$$

kde:	a	vzdálenost mezi laserem a CCD senzorem
	b	vzdálenost od překážky
	β	úhel změřený CCD senzorem.

Dosah limituje jak dosah laseru, tak i velikost plochy čočky CCD senzoru. Určité typy tohoto senzoru umožnují nejen měřit vzdálenost od objektu, ale i pokud se jedná o homogenní průhledné materiály (například sklo), tak dokáží změřit i tloušťku tohoto materiálu. Tato funkce nastane pouze tehdy, když CCD senzor dokáže vyhodnotit dva příchozí paprsky.

Obecné vlastnosti triangulační metody

Čím větší je rozsah měřené vzdálenosti, tím je menší linearita a přesnost měření. Z toho důvodu nastavujeme optiku přijímače. Pro zpracování dat CCD snímače se používá velice rychlý signálový procesor (DSP), který zpracuje data a buď analogovou nebo digitální hodnotu pošle po sběrnici cílovému zařízení například PLC.

Výkon laserového světla je většinou okolo $1\,mW$, vlnová délka červeného paprsku se pohybuje zhruba od 650 do $670\,nm$ [12].

Triangulační metoda je čím dál oblíbenější a rozšířenější, což snižuje cenu senzoru.

2.5 LiDAR

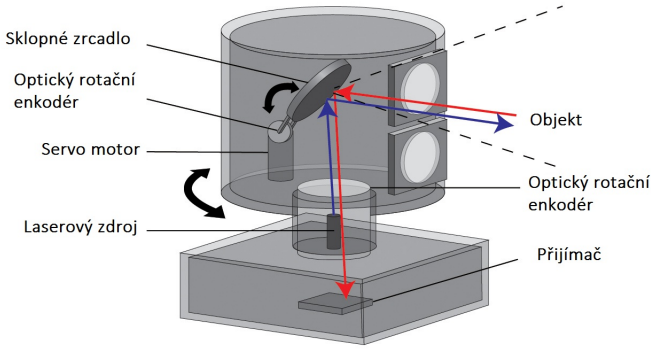
Lidar je technologie bezdrátového snímání vzdálenosti. Ozařuje cil paprskem světla obvykle pulzním laserem. Tuto technologii můžeme vidět stále častěji například na autonomních autech od google maps.

Lidarový skenující systém je složen z laserového měřiče vzdálenosti s rotačním skenovacím zrcadlem a různých polohovacích systémů (POS), včetně GPS, Inerciální měřici jednotkou (IMU) a dalších senzorů. Lidar využívá laserového skenování ve 2D pro vytvoření 360 stupňového panoramatického snímku v horizontální (azimuth) rovině a ve vertikálním zorném poli (FOV) až 40 stupňů. Laserový skener na obrázku 9 využívá metodu návratu (*time-of-flight*) pro každý impulz k určení rozsahu vzdálenosti (*slant*) od objektu. Souřadnice x, y z každého bodu jsou vypočteny a zaneseny do souřadného systémů pomocí pozice a orientace skeneru, to je úhel skenovacího zrcadla a vzdálenost k objektu. Kolekce těchto bodu se nazývá mrak bodů (*point cloud*). Takový mrak bodů lze přímo vizualizovat ve 3D a může být i barevný, jsou-li k dispozici fotogrammetrické informace ve formě digitálních statických snímků fotoaparátů. LiDAR skenery jsou mnohem účinnější než tradiční laserové měřiče vzdálenosti, které měří pouze jeden bod v čase [14].

Obrázek 9: Ukázka LiDAR skenování [14].

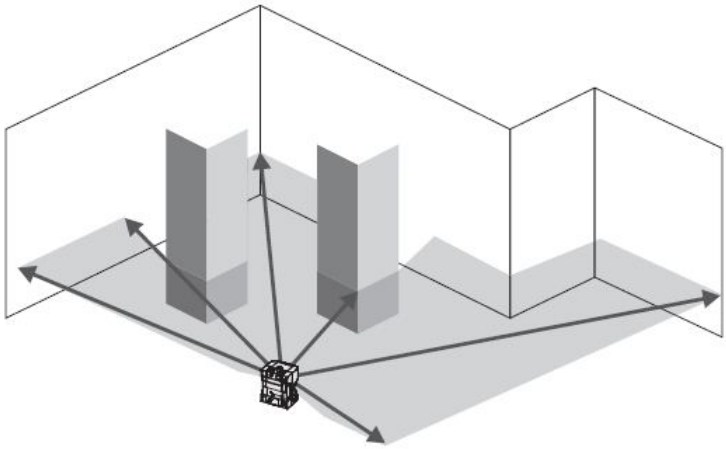
V lidarech se používají dva základní typy uspořádání zrcadel, naklápěcí nebo polygonální. Naklápěcí (*titl*) zrcadla jsou nejčastější, i když oba typy používají rotační snímače pro uzavření smyčky zpětnovazebního řízení skenovaného obrazu. Ve většině starších konstrukcí je rotační enkodér namontován na hřídel zrcadla vedle elektromagnetického pohonu, který slouží k naklápění zrcadel v protisměru hodinových ručiček kolem osy naklonění. Naklánění zrcadel je zodpovědné

za vertikální FOV snímky zobrazeného na obrázku 10. Panoramatické získávání 360 stupňů dat je výsledkem otáčení pouzdra systému LiDAR okolo základny. Výrobci často vyžadují, aby toto pouzdro systému rotovalo vyšší rychlostí než 1000 RPM za účelem zvýšení rozlišení a přesnosti 3D obrazu. To vyžaduje další motor pro otáčení tělesa kolem základny a další rotační enkodér pro přesné řízení zpětné vazby. Nejdokonalejší (*State-of-the-art*) laserové skenery používají matice páru laserových diod a fotodiod na zvýšení rozlišení point cloudu a snížení množství pohyblivých částí. Výsledek toho systému je, že je možné použít pouze jeden rotační enkodér a to na základně systému lidaru. Přesnost enkodéru je kritická pro přesnost systému, neboli každá úhlová chyba může mít za následek zkreslení obrazu. Momentálně nejlepší LiDAR skenery, které se montují na auta, disponují dosahem 120 m a dosahují úhlového rozlišení nižšího než $\pm 0,1$ stupňů. Chyba měření vzdálenosti je menší jako 2 cm na 100 m [14].



Obrázek 10: Schéma LiDAR optiky a enkodérů [14].

V této práci bude simulován pouze 2D lidar, který vytváří horizontální snímek v rozsahu 270 stupňů, dosahu přibližně 2 metry a chybou menši než 10 cm. Princip takového lidaru je zobrazen na obrázku 11.



Obrázek 11: Ukázka principu 2D lidaru.[15]

3 Existující lokalizační metody a algoritmy

3.1 Lokalizace obecně

Lokalizace mobilních robotů je jeden ze základních problémů řízení robotů. Bez lokalizace bychom nemohli dosáhnout autonomního řízení mobilních robotů a ani dalších klíčových úloh, které robot[3] může vykonávat.

V praxi můžeme vidět využití lokalizačních algoritmů například u GPS navigace, kdy pomocí družic na oběžné dráze určíme polohu vozidla, nebo na různých robotických soutěžích, kde je za úkol dostat se z bludiště a podobně.

Lokalizační algoritmy můžeme dělit hned do několika kategorií, a to jsou:

- pravděpodobnostní lokalizace
- relativní lokalizace
- absolutní lokalizace.

3.2 Taxonomie lokalizačních technik

Individuální lokalizační techniky se liší podle jejich určení, způsobilostmi, nezbytnými dispozicemi k vlastnímu fungování a dalšími vlastnosti. Dělí se podle charakteru měření, typu řešeného problému, možnosti lokalizační rutiny ovládání robotu a v neposlední řadě podle dynamiky okolního prostředí [5].

3.3 Relativní a Absolutní lokalizace

- **Relativní lokalizace** slouží k odhadu nebo měření relativní změny polohy robotu, jako je například otočení nebo posun robotu oproti původní pozici. K této problematice se používá především odometrie[4]. Nevýhodou této lokalizace je, že při dlouhodobém používání se snižuje přesnost odhadu polohy [5].
- **Absolutní lokalizace** slouží k odhadnutí pozice v prostoru bez znalosti posunu nebo rotace robotu. Ačkoliv není tato lokalizace zatížena akumulující se chybou, což je její hlavní výhoda, je výpočetně náročnější, a je tedy prováděna s menší frekvencí než lokalizace relativní [5].

[3] Přejaté slovo do většiny jazyků na světě, které použil poprvé Karel Čapek ve hře R.U.R [1]

[4] Původně řecké slovo složeno z hodos (cestovat, cesta) a metron (měřit). Poskytuje informaci o změně polohy nebo orientace robotu.

3.4 Pravděpodobnostní lokalizace

Odhaduje polohu robotu podle veškerých měření absolutních, relativních i jeho poslední pozice. Odhad této pozice není nikoliv jediný bod v prostoru ale místo, kde by se pravděpodobně mohl nacházet. Pozici robotu určuje váha pravděpodobnosti jednotlivých pozic [5].

Pokud uvažujeme o robotech, kteří se pohybují po zemi (například pásové nebo kolové), určujeme polohu l pomocí x, y a orientace θ, viz vztah (3).

$$l = (x, y, \theta), \tag{3}$$

kde: x, y souřadnice polohy robotu
θ úhel natočení robotu.

3.4.1 Metoda Monte Carlo

Základní myšlenka této metody vznikla již v 70. letech, kdy ji objevil J. E. Handschinem [6]. Během dvaceti let byl však nezávisle tento objev znovuobjeven a použit k lokalizaci robotu [7].

Metoda MC je stochastická, využívá pseudonáhodná čísla ke zjištění výsledku. Je to velmi efektivní metoda pro výpočet vícerozměrných určitých integrálů. K tomu, aby tato metoda byla funkční, je potřeba mít kvalitní generátor čísel. V této práci budeme tuto metodu používat zejména k určení polohy robotu, tzv. lokalizace Monte Carlo MCL, a je popsána níže.

Nejčastější využití metody Monte Carlo, dále označovaná jako metoda MC, je vyčíslení Ludolfova čísla π. Jako příklad budeme mít čtverec, který má délku strany $r = 1$, uvnitř čtverce vykreslíme čtvrtinu kružnice o poloměru $r = 1$. Do tohoto prostoru budeme náhodně rozmisťovat částice (body). Z matematiky víme, že obsah čtverce je dán vzorcem (4) a obsah čtvrtiny kruhu vztahem (5). Po dosazení dostaneme vztah (6). Pokud tedy za $S_{\check{c}k}$ dosadíme částice, které jsou uvnitř čtvrtiny kruhu, a za $S_{\check{c}}$ dosadíme celkový počet částic, dostaneme přibližnou hodnotu π podle vztahu (7). Pro lepší představu byl vytvořen program v C#, viz obrázek 12, kdy u obrázku a) kde je 50 vzorku vyšlo 2,96 což je velmi nepřesný odhad čísla π naopak u obrázku b) a c) kde je 500 a 5 000 vzorků můžeme vidět že se odhad (3,264 a 3,148) přibližuje skutečné hodnotě čísla π.

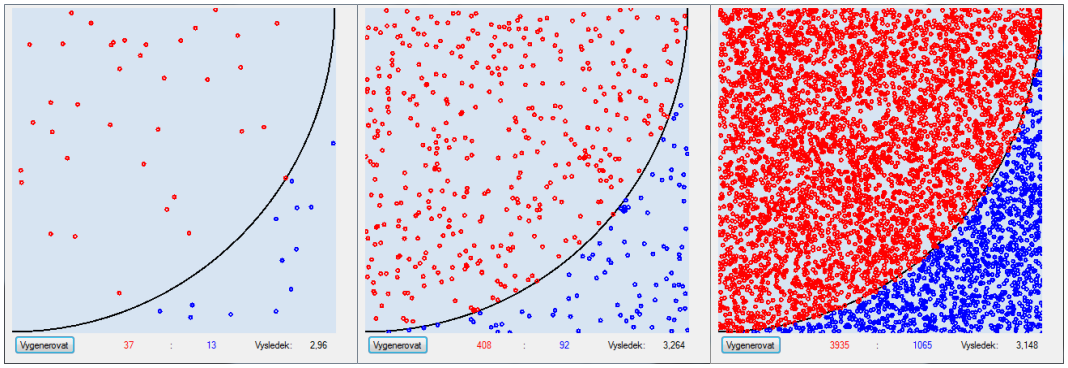
$$S_{\check{c}} = r^2 \tag{4}$$

$$S_{\check{c}k} = \frac{\pi \cdot r^2}{4} \tag{5}$$

$$S_{\check{c}k} = \frac{\pi \cdot S_{\check{c}}}{4} \tag{6}$$

$$\pi \approx \frac{n_{\check{c}k} \cdot 4}{n_{\check{c}}}, \tag{7}$$

kde:	r	délka strany čtverce a poloměr kruhu
	$S_{\text{č}}$	obsah čtverce
	$S_{\text{čk}}$	obsah čtvrtiny kruhu
	$n_{\text{č}}$	počet částic ve čtverci
	$n_{\text{čk}}$	počet částic čtvrtiny kruhu.



Obrázek 12: Ukázka Metody Monte Carlo: a) 50 vzorků, b) 500 vzorků, c) 5000 vzorků.

3.4.2 Lokalizace Monte Carlo

Po seznámení se s metodou MC ukážeme princip použití pro lokalizaci mobilních robotů. Základem, jak již bylo zmíněno, je náhodné rozmístění množiny m částic, které v tomto případě nereprezentují jen polohu, ale i váhu příslušné částice, viz (8). Tato váha, kterou značíme w_i, reprezentuje důležitost částice. Částice s největší váhou má vyšší pravděpodobnost, že odpovídá skutečné pozici robotu.

$$m[i] = (w_i, l), \tag{8}$$

kde:	m	množina částic
	w_i	váha částice
	l	poloha částice podle vztahu (3).

Celkový počet částic n je omezen pouze výpočetním systémem a optimalizací. Většinou se pohybuje okolo stovek až do několika tisíců. Většinou platí, že čím víc vzorků, tím je robot schopný rychleji lokalizovat svou pozici.Polohu robotu v prostoru určuje jak hustota částic v určitém prostoru, tak i jejich váha. Tyto dva parametry dohromady určují hustotu pravděpodobnosti výskytu robotu, který je přibližně vyjádřen v celém prostoru pomocí odhadu polohy $Bel(l)$ dle vzorce (9).

$$Bel(l) \approx \sum_{i=0}^{n} w_i \cdot \delta \cdot (l - l_i), \tag{9}$$

kde:	δ	diskrétní Diracův impulz
	w_i	váha částice
	l	poloha částice
	l_i	poloha částice.

Nyní jsme si popsali vlastnosti jednotlivých částic a nyní bude rozebrán kompletní algoritmus MCL. V jiných oborech je MCL známý pod názvem *částicový filtr* (*particle filter*) [11], *kondenzační algoritmus, bootstrap filters, interacting particle approximations* nebo *survival of the fittest* [8].

Hlavní podstatou této metody byla vhodná aproximace důvěry *Bel(l)* váženou množinou částic tak, aby diskrétní distribuce definované částice skutečně odpovídala výchozí spojité distribuci [9].

V počátku je rozložení důvěry představující jednotvárné roztřídění množiny částic m, kdy všechny částice dostanou váhu, rovné jedné-viz (10). Díky tomuto způsobu reprezentace pomocí vážených vzorků můžeme vyjádřit pevný odhad polohy $Bel(l)$ více způsoby, a to buď většími váhami patřičných vzorků nebo také větší hustotou vzorků v dané části prostoru. V průběhu mohou nastat případy, kdy váhy všech vzorků budou stejné, například všechny vzorky budou mimo dosah dálkových senzorů. Poté bude dán odhad polohy pouze rozdělením částic v prostoru. V druhém případě může nastat situace, že částice se rozmístí pravidelně do prostoru a odhad se bude tím pádem lišit pouze váhami jednotlivých částic. Reálná reprezentace u praktického použití v MCL by se měla pohybovat většinou mezi těmito dvěma extrémy.

$$\sum_{i=0}^{n} w_i = 1 \tag{10}$$

Celkový algoritmus MCL lze rozdělit do čtyř základních kroků:

- inicializace
- predikce
- korekce
- převzorkovaní.

Inicializace

U inicializace dochází k již zmíněnému jednotvárnému rozložení důvěry, tudíž všechny částice mají zpočátku váhu w_i rovno jedné. Dále dochází k náhodnému rozložení polohy částic v prostoru.

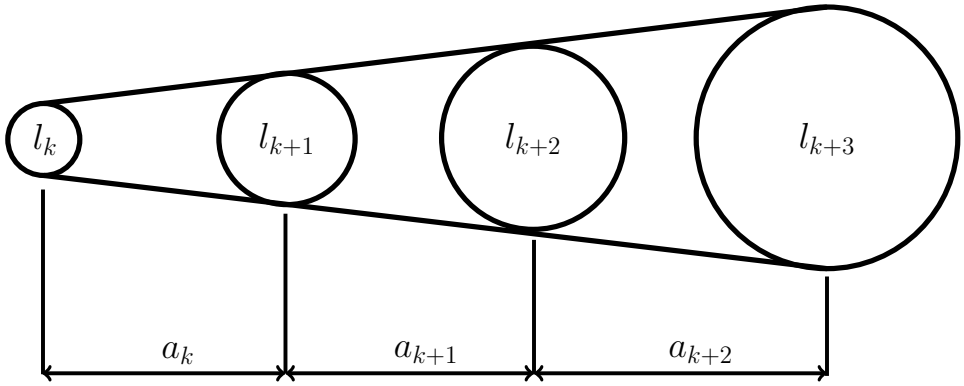
Predikční krok

U tohoto kroku si nejprve musíme definovat pravděpodobnostní pohybový model nebo akční model (anglicky *motion model* resp. *action model*). Tento model slouží k vypočtu $Bel^{-}(l_k)$ a

popisuje, jakým způsobem určitá akce a_{k-1} změní polohu robotu z původní pozice l_{k-1} na novou l_k. V tomto případě jde o posun robotu na základě změřených dat z relativních senzorů (odometrie). Tento model můžeme popsat podmíněnou pravděpodobností podle vzorce (11).

$$P(l_k \mid l_{k-1}, a_{k-1}) \tag{11}$$

V predikčním kroku dochází k aktualizaci odhadu polohy na základě posunu robotu, protože nový odhad polohy předpovídá na základě předešlého odhadu a předešlých informací z relativních dat o změně polohy robotu. Tento vytvořeny odhad pomocí predikčního kroku nazýváme apriorní odhad polohy (anglicky *prior beleif*). K Výpočtu apriorní polohy je zapotřebí již definovaný pravděpodobnostní model. Váha w_i se u predikčního kroku nemění, tento krok je znázorněn na obrázku 13.



Obrázek 13: Akumulace chyby při použití predikčního kroku.

Korekční krok

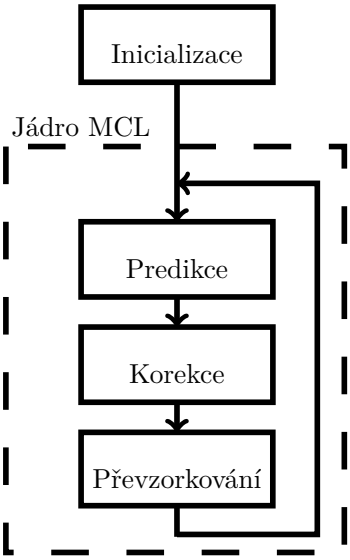
Tento krok využívá informace o absolutní poloze robotu a podle těchto informací aktualizuje množinu částic na základě podobnosti těchto měření, které odpovídají pozici jednotlivých částic. Podle podobnosti upraví váhu jednotlivých částic. Způsob rozdělení vah pomocí normalizace jednotlivých absolutních měřeni muže být prováděn různými způsoby, nejlépe podle Gaussovského rozložení.

Odhad pozice vytvořený v tomto kroku se označuje jako aposteriorní odhad polohy (anglicky *posterior beleif*).

Převzorkování

Účelem tohoto kroku je v podstatě eliminace vzorku, které mají nízkou váhu. Naopak vzorky s váhou velkou rozdělit na vzorku několik. Tím vznikne nová množina vzorků M. Tento krok zajistí, že se výpočetní výkon nebude zabývat vzorky s nízkou váhou, ale soustředí jej na vzorky s váhou vysokou.

Všechny tyto kroky kromě inicializace se neustále opakují, dokud se hustota částic nesjednotí do jednoho bodu v daném prostoru, který bude reprezentovat pozici robotu. Blokový diagram tohoto algoritmu je vyobrazen na obrázku 14.



Obrázek 14: Algoritmus MCL.

4 Prostředky pro tvorbu simulací (C++, C# , Matlab)

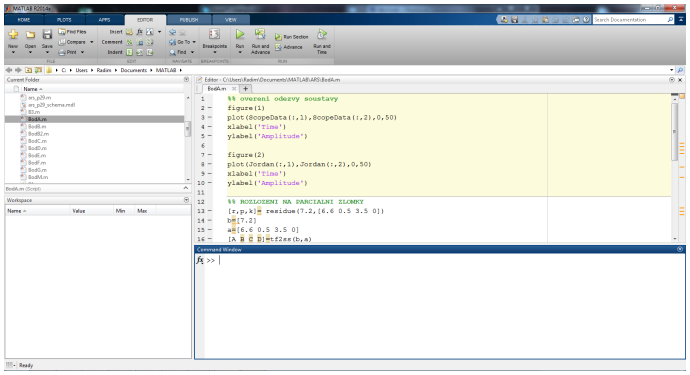
Pro tvorbu simulací je potřeba si uvědomit, jak náročnou simulaci budeme chtít provádět a také v jakém prostředí bychom ji chtěli realizovat. V dnešní době je dostupná široká škála různých prostředí pro tvorbu simulací, které se od sebe odlišují například způsobem psaní kódu. Pro způsoby psaní existuje celá řada programovacích jazyků nejen textových, ale i grafických[5]. Z tohoto nepřeberného množství jsem si vybral tři velmi efektivní nástroje, které jsou popsány v následujících bodech.

4.1 MatLab

Původní účel Matlabu byl pro přístup k matematickým knihovnám. Matlab vlastní firma The MathWorks. Matlab je jak název programu, tak i programovací jazyk. Jeho základní datový typ je matice. Mezi další patří např. pole reálných i komplexních čísel, ale můžeme si i sami nadefinovat vlastní datové typy [2].

Prostředí Matlab je určeno pro řešení matematických výpočtů, ale i pro tvorbu různých výpočetních algoritmů. Pro tyto účely nabízí možnosti použití opravdu mocných nástrojů, které „umožňují uživatelům manipulovat a analyzovat data pomocí obrázků, grafů, tabulek a algoritmů“. Vývojové prostředí Matlabu je znázorněno na obrázku 15.

Toto prostředí je vytvořeno pro velice rychlý návrh výpočetních algoritmů, tudíž je velmi efektivní. Na druhou stranu je Matlab jeden z nejdražších výpočetních programů na trhu.

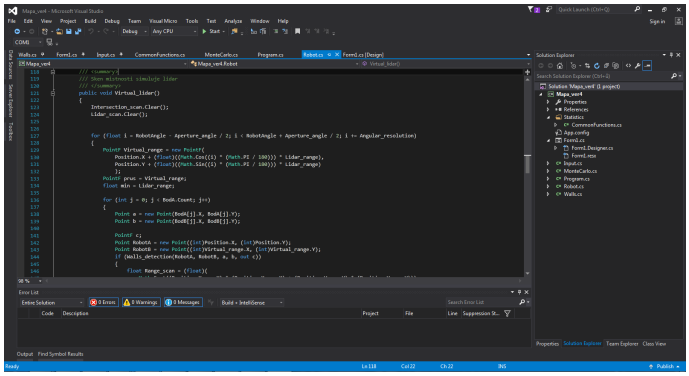


Obrázek 15: Ukázka vývojového prostředí MatLab.

4.2 C#

C# tento programovací jazyk je zařazen do skupiny objektových programovacích jazyků a patří mezi vysokoúrovňové jazyky. Je založený na platformě .NET Framework vyvinutý firmou Microsoft. Jeho syntaxe je odvozena z jazyku C, ale je doplněn o řadu vylepšení v podobě funkcí pro

[5]Například LabVIEW,MatLab simulink



Obrázek 16: Ukázka vývojového prostředí Visual Studio.

rychlejší vývoj aplikací. Pro programování v jazyce C# vyvinul Microsoft prostředí Visual Studio (viz obrázek 16). Toto studio obsahuje kromě kompletní výbavy pro ladění programů také grafické prostředí, kde můžeme vkládat například tlačítka, text a podobně.

Oproti Matlabu má tento jazyk výhodu v tom, že provádění instrukcí kódu je podstatně rychlejší, máme více možností vytvoření dané aplikace a nejsme omezeni na rozsah funkcí daného prostředí. Dále můžeme tvořit webové aplikace nebo aplikace pro mobilní zařízení. Mezi jeho další velkou výhodu patří, že po dokončení aplikace můžeme vytvořit *.exe* soubor, který můžeme spustit bez potřeby nainstalovaného prostředí pro práci s jazykem C#, a to může být velmi ekonomicky výhodné. Nevýhodou je větší časová náročnost na vývoj aplikací oproti Matlabu, protože většinu funkcí si musíme sami naprogramovat.

4.3 C++

Tento jazyk stejně jako C# můžeme programovat objektově, ale je možné programovat i neobjektově. Vyvinul se z jazyka C. C++ a vytvořil ho Bjarne Stroustrup. Je to nejrychlejší jazyk z předchozích dvou, ale také i nejsložitější. Nenabízí žádné speciální funkce. Pokud chceme nějakou funkci, tak ji musíme vytvořit pomocí tříd nebo metod. Syntaxe programování nejen vychází z jazyka C, ale podporuje všechny příkazy [3]. K programovaní v C++ je velká dostupnost různých editoru, navíc tento jazyk můžeme použít přímo k programovaní mikroprocesoru, mikrokontroléru, embedded systému[6] nebo malých počítačů jako například Raspberry Pi. Jelikož v dnešní době je pro tento jazyk hodně i kvalitně zpracovaných editoru, tak můžeme tvořit aplikace bezplatně.

Mezi oblíbené editory patří již zmiňované Visual Studio od firmy Microsoft nebo také Open-source editor Code Blocks IDE.

[6] Jednoúčelové zařízení obsahující zabudovaný počítač, který ovládá zařízení.

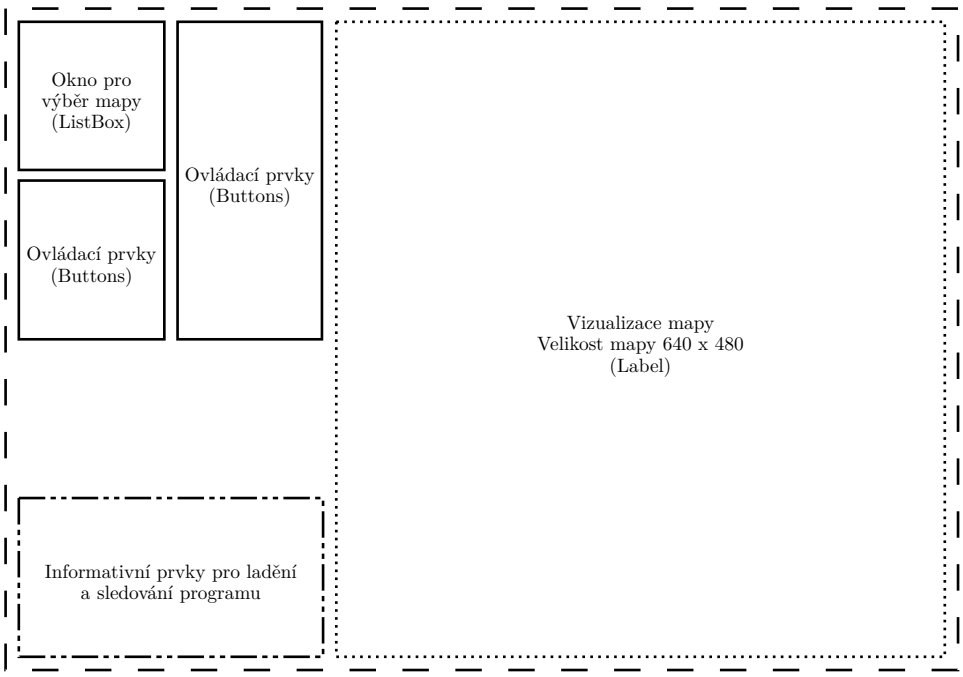
5 Návrh vlastního řešení

Prvním bodem praktické části mé bakalářské práce je vytvoření simulačního programu pro realizaci simulace lokalizačního algoritmu Monte Carlo. Pomocí teoretických znalostí získaných v předchozích kapitolách byl implementován program simulující chování mobilního robotu.

Realizace programu je provedena ve vývojovém prostředí Visual Studio pomoci jazyka C#, toto prostředí je již popsáno v předchozí kapitole. Postup vytvoření programu je popsán v následujících podkapitolách.

5.1 Koncept programu

Před začátkem programování bylo potřeba určit, jak samotné simulační prostředí bude vypadat. Dále co vše bude obsahovat, například umístění tlačítek, plochy pro vizualizaci, seznam pro výběr mapy a informativních panelů. Pro tento účel byl použit projekt ve Windows Froms Applicaton, který vytvoří spustitelnou okenní aplikaci pro Windows. V této aplikaci je možnost vybrat objekt z panelu nástrojů (Toolbox), který obsahuje prvky jako tlačítko (Button), textové pole (Label), vykreslovací pole (panel), seznam položek (ListBox) a další. Tyto prvky velmi urychlují vytvoření programu. Koncept je znázorněn na obrázku 17.



Obrázek 17: Koncept simulačního programu.

5.2 Načtení mapy ze souboru

Pro potřeby simulace byl vytvořen balík map, který je uložen ve složce Maps ve formátu textového souboru *.txt*. Tento soubor obsahuje jednotlivé stěny mapy ve formátu

$$x_{\mathrm{As}}; y_{\mathrm{As}}; x_{\mathrm{Bs}}; y_{\mathrm{Bs}}; w_{\mathrm{s}},$$

kde:	$x_{\mathrm{As}}, y_{\mathrm{As}}$	začátek stěny
	$x_{\mathrm{Bs}}, y_{\mathrm{Bs}}$	konec stěny
	w_{s}	šířka stěny.

K načtení mapy do programu je využit nástroj listBox, který při spuštění programu ukáže dostupné mapy. Po kliknutí na libovolnou mapu dojde k nahrání a uložení mapy do třídy List (Kolekce, která umožnuje v průběhu programu mazat nebo nahrávat jednotlivé prvky) [10]. Tato operace je ukázána ve výpise 1. Mapa se ihned po nahrání graficky zobrazí na panelu.

```
using (StreamReader sr = new StreamReader(new FileInfo(Application.ExecutablePath).DirectoryName
     + @"\Maps\ " + Maps))
  {
    string s;
    while ((s = sr.ReadLine()) != null)
    {
      string[] ArrayCoordinates = s.Split(';');
      BodA.Add(new Point(int.Parse(ArrayCoordinates[0]),int.Parse(ArrayCoordinates[1])));
      BodB.Add(new Point(int.Parse(ArrayCoordinates[2]),int.Parse(ArrayCoordinates[3])));
      WallsWidth.Add(int.Parse(ArrayCoordinates[4]));
    }
  }
```

Výpis 1: Metoda pro čtení ze souboru a následné parsování.

5.3 Vytvoření modelu robotu

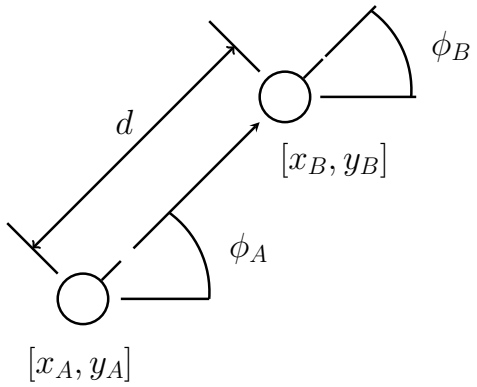
Při vytváření třídy robotu bylo třeba vymyslet několik věcí, aby se robot choval co nejvěrněji reálnému robotu. Mezi jeho vlastnosti patří detekce stěny, odometrie, laserový dálkoměr, grafické vykreslení.

5.3.1 Posun robotu

První krok při tvorbě modelu byl posun a otáčení robotu. K vyřešení tohoto problému bylo potřeba použít vzorec (12).

$$\begin{aligned} \phi_B &= \phi_A \\ x_B &= x_A \cdot \cos(\phi_A \cdot \tfrac{\pi}{180}) \cdot d, \\ y_B &= y_A \cdot \sin(\phi_A \cdot \tfrac{\pi}{180}) \cdot d \end{aligned} \tag{12}$$

kde:	d	délka posunu robotu
	x_A, y_A, ϕ_A	původní souřadnice
	x_B, y_B, ϕ_B	souřadnice po přesunutí.



Obrázek 18: Model posunu.

K dosažení reálné simulace bylo zapotřebí zohlednit také chyby odometrie robotu jako například proklouznutí kola nebo nepřesné měření při otáčení. Aby tato chyba byla reálná, ale také i dostatečně náhodná, byla použita upravená Gaussova funkce. Do této funkce vstupují dva parametry σ a μ. σ znázorňuje maximální odchylku od střední hodnoty μ. Jelikož se chyba akumuluje s ujetou vzdáleností, tak je σ dopočítávána z délky posunutí robotu. Implementaci této funkce můžeme vidět níže ve výpise 2. Z této metody dostaneme náhodnou odchylku, kterou následně přičteme k souřadnicím robotu 3. Velikost chyby je možno nastavit v konstruktoru třídy Robot.

```
public int GaussFun(Random rnd, float sigma, int mean)
  {
    float rnd1 = (float)rnd.NextDouble();
    float rnd2 = (float)rnd.NextDouble();

    float x1 = 2 * rnd1 / 1 - 1;
    float x2 = 2 * rnd2 / 1 - 1;
    float w = (x1 * x1 + x2 * x2) / 2;
    float M = (float)(x1 * Math.Sqrt(-2 * Math.Log(w) / w) * sigma);
```

```
    return (int)M;

}
```

Výpis 2: Implementace upravené Gaussovy funkce.

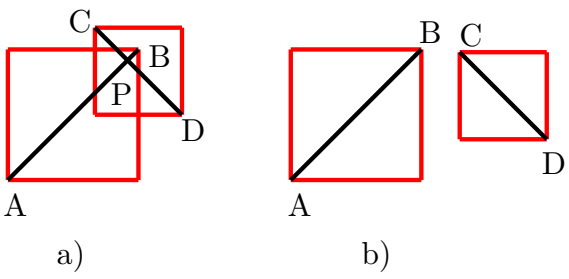
```
Position.X = Position.X + G.GaussFun(rnd, sigma_x, (int)dx);
Position.Y = Position.Y + G.GaussFun(rnd, sigma_y, (int)dy);
RobotAngle = RobotAngle + G.GaussFun(rnd, sigma, 0);
```

Výpis 3: Přičtení výsledné chyby k posunu.

5.3.2 Detekce stěny

K tomu, aby robot dokázal určit vzdálenost od stěny a následně i vytvořit sken pro simulaci lidaru, je potřeba použít metodu, která dokáže najít průsečíky dvou přímek. K tomuto účelu byla použita metoda nejmenších čtverců pro hledání průsečíku robotu a stěny. Tato metoda funguje na základě čtyř souřadnic, mezi které patří počáteční souřadnice stěny, konečné souřadnice stěny, souřadnice robotu a imaginární dopočítaný bod dosahu lidaru. Dále metoda čtverců umožnuje po dosazení parametru zjištění, zda se přímky protínají, a pokud ano, tak i přibližnou polohu průsečíku. Princip teto metody je znázorněn na obrázku 19. V případě, že je průsečíků více, tak je vybrána kratší vzdálenost robotu a průsečíku.

Tato operace je vykonána pro každý paprsek lidaru v určitém rozptylu, který je možno navolit v konstruktoru třídy Robot. Ke každému paprsku je přidán šum pro důvěryhodnější měření opět pomocí Gaussovy funkce zmíněné výše.



Obrázek 19: a) Stav kdy metoda nalezne průsečík P b) Stav kdy metoda vyhodnotí (false).

```
private bool Walls_detection(Point a, Point b, Point c, Point d, out PointF P)
 {
   P = new PointF(-20, -20);
   float denominator = ((b.X - a.X) * (d.Y - c.Y)) - ((b.Y - a.Y) * (d.X - c.X));
   float numerator1 = ((a.Y - c.Y) * (d.X - c.X)) - ((a.X - c.X) * (d.Y - c.Y));
   float numerator2 = ((a.Y - c.Y) * (b.X - a.X)) - ((a.X - c.X) * (b.Y - a.Y));

   if (denominator == 0) return numerator1 == 0 && numerator2 == 0;
```

```
  float r = numerator1 / denominator;
  float s = numerator2 / denominator;

  P.X = a.X + (r * (b.X - a.X));
  P.Y = a.Y + (r * (b.Y - a.Y));

  return (r >= 0 && r <= 1) && (s >= 0 && s <= 1);
}
```

Výpis 4: Metoda pro zjištění průsečíku.

5.3.3 Grafické znázornění robotu

Pro grafické zobrazení bylo použito vektorové grafiky. Objekt symbolizující robota se skládá s kruhu a přímky, která znázorňuje úhel natočení robotu.

6 Implementace vybraného lokalizačního algoritmu

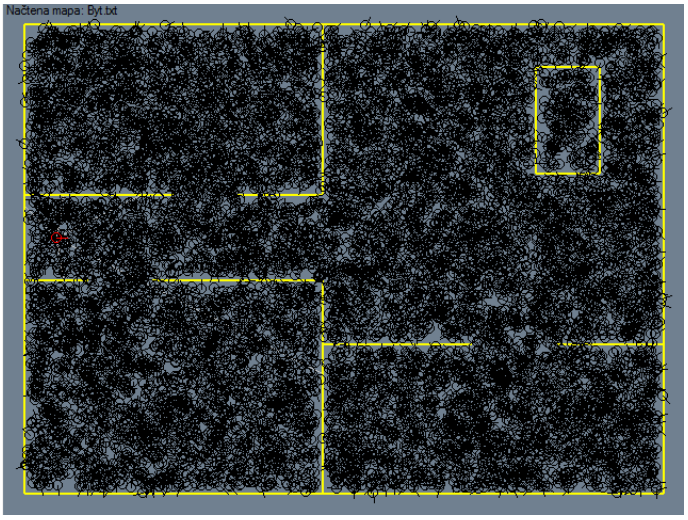
Po vytvoření simulačního nástroje popsaného v kapitole 5 bylo možné implementovat příslušný lokalizační algoritmus. Z důvodu velké popularity a také zajímavosti metody byla vybrána stochastická lokalizace Monte Carlo, jejíž princip je popsán v kapitole 3.4.2. Nyní bude podrobně popsaný způsob implementace vybraného lokalizačního algoritmu.

Jak již bylo zmíněno v kapitole 3.4.2, MCL má čtyři základní operace: inicializace, predikce, korekce a převzorkování, z čehož se inicializace vykoná pouze při prvním kroku MCL. Ostatní operace se provádějí stále dokola.

Pro lepší přehlednost programu byla k implementaci algoritmu MCL vytvořena třída *MonteCarlo.cs*. Tato třída obsahuje všechny potřebné metody až na predikční krok, který již byl implementován přímo do třídy *Robot.cs*.

6.1 Inicializace

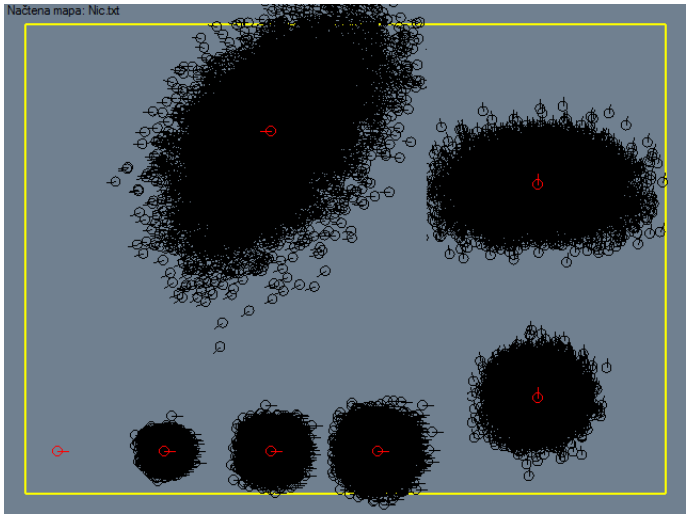
Řešení inicializace je provedeno v metodě *Inicialization*. Do této metody vstupuje pouze jeden parametr a to pole instance třídy *Robot* pod názvem *Particles*, které obsahuje jednotlivé částice robotu. Funkci této metody jsou dva kroky. První je nastavení počáteční pozice jednotlivých částic robotu. K tomu účelu je využita metoda *Spawn_Robot* třídy Robot, která náhodně rozmístí jednotlivé částice po mapě. Druhý krok je nastavení váhy $w_i = 1$ pro každou částici. Simulace po prvním kroku je naznačena na obrázku 20, kde je červený objekt hlavní robot (anglicky *leader follower*) a ostatní pravděpodobné pozice hlavního robotu (anglicky *follower*). Po tomto kroku následuje predikční krok.



Obrázek 20: Inicializační krok (10000 částic).

6.2 Predikce

U predikčního kroku dochází, jak již bylo zmíněno v kapitole 3.4.2, k aktualizaci polohy robotu na základě jejich posunu, kdy je dopočítána jistá odchylka kvůli akumulaci chyb u relativních senzorů. Tato odchylka je popsaná v kapitole 5. Jednotlivé kroky s využitím pouze predikce můžeme vidět na obrázku 21, kde vidíme akumulaci chyby a zvyšující se nejistotu skutečné pozice robotu. Po přesun robotu na novou pozici je vykonána korekce.



Obrázek 21: Predikční krok.

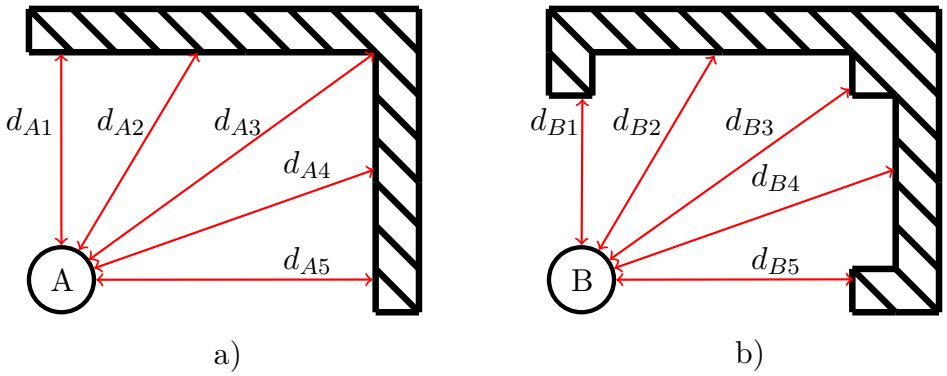
6.3 Korekce

Jak již víme, tak tento krok nevyužívá relativní senzory, ale senzory absolutní. V podstatě každá částice vytvoří virtuální sken lidaru a porovnává jej s hlavním robotem. Pro princip porovnávání skenu jsem vytvořil obrázek 22. Na tomto obrázku vidíme roboty A a B. Robot A bude symbolizovat hlavní robot a robot B pravděpodobného. $d_{A1} - d_{A5}$,$d_{B1} - d_{B5}$ reprezentují délky jednotlivých paprsků lidaru. Jako první krok se provede suma podle vzorce (13). Z tohoto vzorce dostaneme nenormalizovanou váhu částice B, W_B. Z nenormalizovaných vah W_i je vybrána maximální hodnota váhy $W_{\max}$, z této hodnoty pomocí vzorců (14) získáme koeficienty W_{10} a W_{30}, které mají deseti a třiceti procentní hodnoty $W_{\max}$. Pomocí těchto tří proměnných ($W_{\max}$,W_{10},W_{30}) vytvoříme tři intervaly viz vzorec (15). Dále do tohoto modelu dosazujeme váhy jednotlivých částic a dostáváme již normalizovanou váhu, která nabývá hodnot od nuly do jedné.

$$W_{\mathrm{B}} = \sum_{i=0}^{n} (d_{\mathrm{Bi}} - d_{\mathrm{Ai}})^2 \tag{13}$$

$$\begin{aligned} W_{10} &= W_{\max} \cdot 0,1, \\ W_{30} &= W_{\max} \cdot 0,3 \end{aligned} \tag{14}$$

kde:	W_B	nenormalizovaná váha částice
	d_{Ai}	délka paprsku lidaru na hlavním robotu
	d_{Bi}	délka paprsku lidaru na pravděpodobném robotu
	$W_{\max}$	je maximální odchylka od skenu
	W_{10}	je 10 % $W_{\max}$
	W_{30}	je 30 % $W_{\max}$.



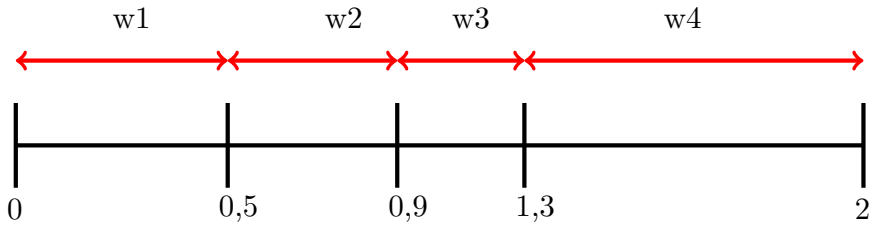
Obrázek 22: Princip skenu lidaru.

$$\langle w_i \rangle = \begin{cases} 1,9 - \frac{W_i}{W_{10}} & \langle 0, W_{10} \rangle \\ 1,2 - \frac{W_i}{W_{30}} & (W_{10}, W_{30} \rangle \\ 0 & (W_{30}, W_{\max} \rangle \end{cases} \tag{15}$$

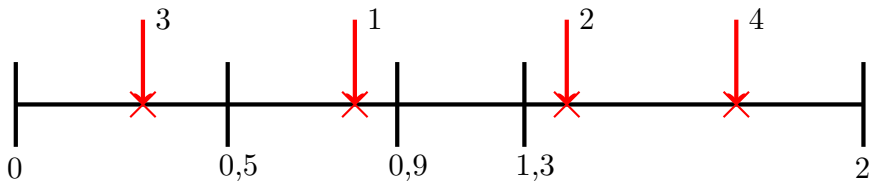
6.4 Převzorkování

Tento krok má za účel, jak již bylo zmíněno v kapitole 3, eliminovat částice s malou váhou a naopak částice s váhou velkou rozdělit na více částic.

Princip funkce tohoto kroku spočívá ve vytvoření úseček o velikosti jednotlivých vah částic a následného seřazení do jedné linky viz obrázek 23 a). Dále se generují náhodna čísla v rozsahu 0 až velikost sumy vah všech částic, výsledná hodnota náhodného čísla označí bod na některé úsečce viz. obrázek 23 b), kde z tohoto obrázku můžeme vidět, že první náhodné číslo ukazuje na úsečku, která náleží částici dva. Tudíž druhá částice překopíruje svou pozici do prvního nového vzorku. Tato operace se opakuje, dokud nemají všechny nové částice novou polohu. Po dokončení dojde k vykreslení nové množiny vzorku na mapu.



a)



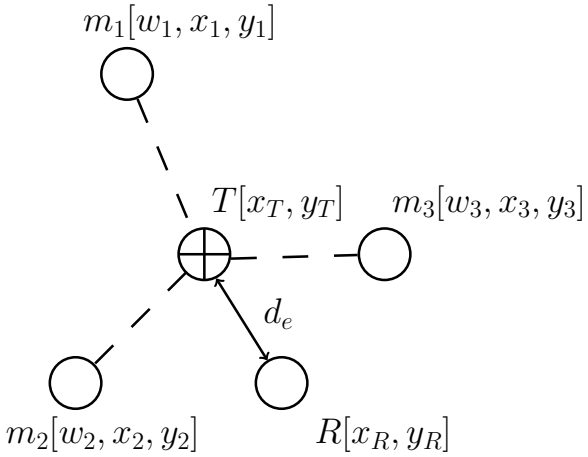
b)

Obrázek 23: Způsob a) seřazení úseček. b) generování náhodých čísel.

7 Testování

V této kapitole otestujeme simulační program, který byl popsán v kapitole 5 a 6. Dále otestujeme chováni simulace při změnách přesnosti senzoru a rychlost lokalizace při různém počtu vzorků.

Pro potřeby testování bylo potřeba vytvořit těžiště T množiny vážených vzorku m pro výpočet jednak odchylky od skutečné pozice d_e, ale také i pro výpočet směrodatné odchylky S. Model pro výpočet je zobrazený na obrázku 24. Pro výpočet těžiště platí vzorec (16), odchylky (17) a směrodatné odchylky (18).



Obrázek 24: Model pro výpočty.

$$x_T = \frac{\sum_{i=0}^{n}(x_i \cdot w_i)}{\sum_{i=0}^{n}(w_i)}, y_T = \frac{\sum_{i=0}^{n}(y_i \cdot w_i)}{\sum_{i=0}^{n}(w_i)} \tag{16}$$

$$d_e = \sqrt{(x_R - x_T)^2 + (y_R - y_T)^2} \tag{17}$$

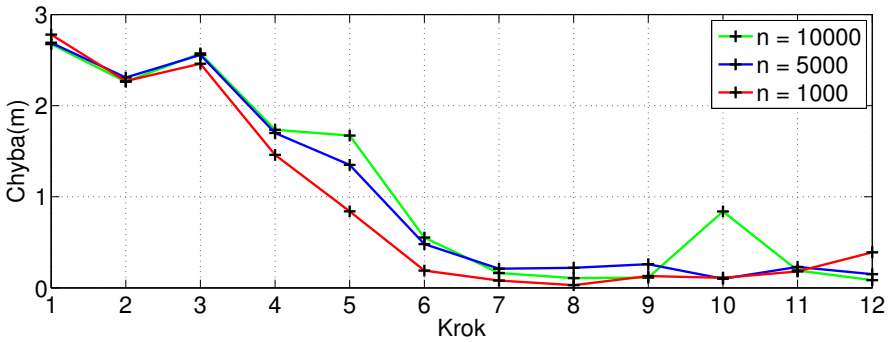
$$S = \sqrt{\frac{1}{n} \cdot \sum_{i=0}^{n}(X_i - X_T)} \tag{18}$$

$$X_i = \sqrt{x_i^2 + y_i^2}, X_T = \sqrt{x_T^2 + y_T^2}, \tag{19}$$

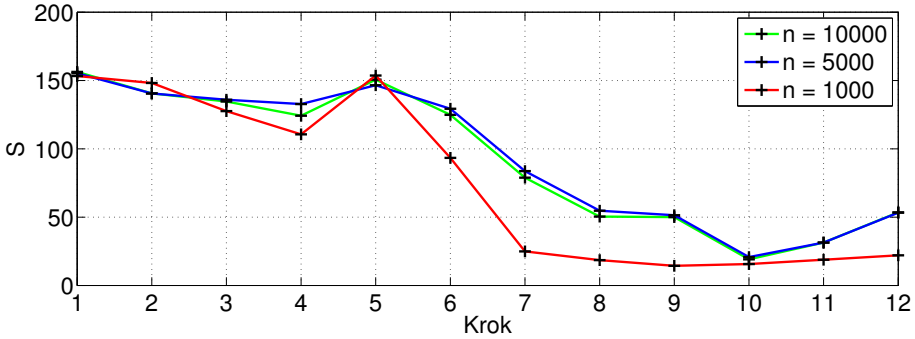
kde:	x_T, y_T	souřadnice těžiště T
	x_i, y_i	souřadnice pravděpodobné pozice částice m_i
	w_i	váha častice m_i
	d_e	vzdálenost těžiště od hlavního robotu
	x_R, y_R	souřadnice hlavního robotu
	S	směrodatná odchylka
	X_i, X_T	moduly pro výpočet směrodatné odchylky.

7.1 Testováni MCL

Průběh simulačního programu pří $n = 10\,000$ částic je zobrazen na sekvenci obrázcích 31 až 42, které jsou umístěny v příloze BP. Dále bylo změřeno více simulaci při různém počtu částic. Tyto simulace jsou vyobrazeny ve dvou grafech, z nichž graf 25 zobrazuje chybu d_e v závislosti na kroku programu. Tato chyba odpovídá vzdálenosti robotu od těžiště. Druhý graf 26 zachycuje maximální rozptyl všech částic v závislosti na kroku programu.



Obrázek 25: Průběh chyby vzdálenosti d_{yrme}



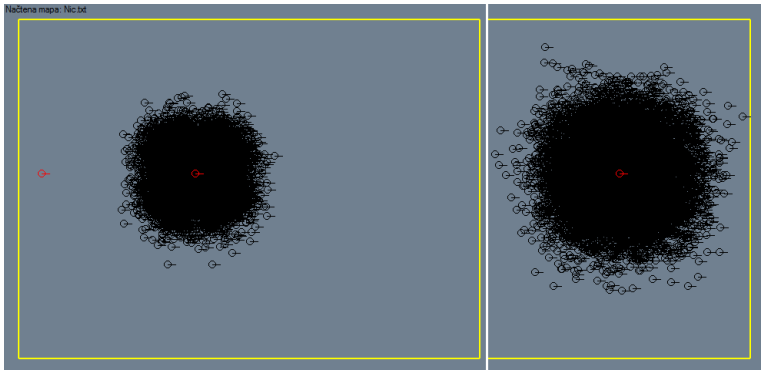
Obrázek 26: Průběh chyby směrodatné odchylky S

7.2 Testováni akumulace chyby relativních senzoru polohy

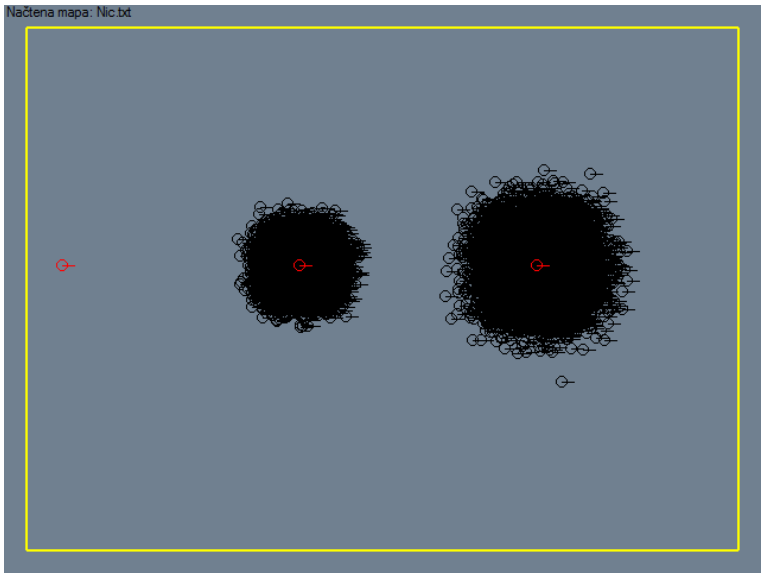
Toto testováni bylo provedeno na základě změny velikosti dopočítávání σ pro Gaussovu metodu viz. 2. Sigma je dopočítávána jako součin vzdálenosti, kterou jsme ujeli, a koeficient T_e. Tyto koeficienty T_e byly nastaveny průběžně na tři testy s velikosti $0,1$, $0,05$, $0,01$.

Průběh simulace akumulace chyby relativních senzorů byl pro jednotlivé velikosti koeficientu T_e zobrazen do tří obrázků 27 až 29.

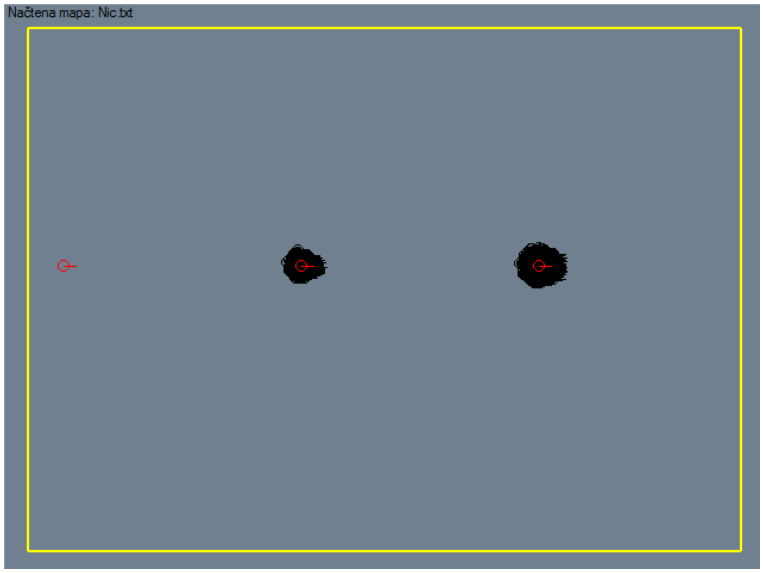
Pro celkové vyhodnocení bylo dopočítáno směrodatnou odchylkou S, kdy S udává maximální velikost rozptylu částic od hlavního robotu. Tento rozptyl je zaznamenán v grafu 30.



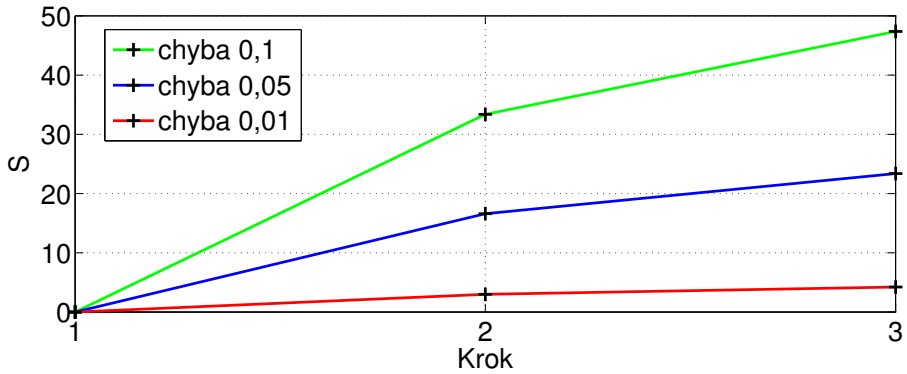
Obrázek 27: Průběh akumulace chyby při chybě 0,1



Obrázek 28: Průběh akumulace chyby při chybě 0,05



Obrázek 29: Průběh akumulace chyby při chybě 0,01



Obrázek 30: Průběh chyby směrodatné odchylky S

8 Zhodnocení dosažených výsledků

Jak si můžeme všimnout, v kapitole 7 konkrétně u podkapitoly 7.1 z grafu 25 je vidět, že více vzorků mělo spíše negativní vliv na přesnost lokalizace. Dále je vidět, že se robot dokázal lokalizovat, nehledě na počet vzorků již v sedmém kroku simulace. Z druhého grafu 26, který zobrazuje maximální rozptyl částic, můžeme také vidět, že více vzorků mělo negativní vliv.

u druhého testovaní v podkapitole 7.2 na obrazcích 27 až 29 můžeme vidět vliv koeficientu T_{e} na simulaci. V případě $T_{\mathrm{e}} = 0{,}1$ vidíme, že se částice rozptýlily do velkého prostoru pravděpodobnosti, naopak u $T_{\mathrm{e}} = 0{,}01$ je tento rozptyl minimální.

9 Závěr

Bakalářská práce se zabývala vývojem simulačního nástroje. Tento nástroj simuluje chování jak relativních senzorů polohy, kterými jsou například enkodéry, tak i chování absolutních senzorů polohy, například LiDAR.

Kapitola 2 se zabývala senzory, které jsou často využívány v lokalizaci mobilních robotů. Tyto senzory byly například pro měření ujeté vzdálenosti (Enkodéry) nebo zjišťování vzdálenosti robotu od překážky (ultrazvuk, Lidar). Kapitola 3 popisovala různé způsoby pro lokalizaci robotů. Tyto způsoby byly například relativní a absolutní lokalizace. Podrobněji se tato kapitola věnovala lokalizaci a metodě Monte Carlo.Kapitola 4 popisovala prostředky pro tvorbu simulací. Konkrétně tři vývojové jazyky a to Matlab, C# a C++. Další text práce se věnoval praktické tvorbě a testování vytvořeného programu. Kapitola 5 se zabývala zhotovením simulačního programu. Konkrétně simulací robotického vozítka a jeho senzorů vzdálenosti a posunu. Kapitola 6 popisovala implementaci jednotlivých kroků lokalizace Monte Carlo. Tyto kroky byly inicializace, predikce, korekce a převzorkování. Kapitola 7 se zabývala testováním průběhu simulačního programu a samotné lokalizace Monte Carlo. V průběhu simulace bylo zjišťováno několik faktorů a to vzdálenost hlavního robotu od těžiště vážených vzorku a maximální rozptyl částic pomocí směrodatné odchylky. V Kapitole 8 byly zhodnoceny dosažené výsledky z průběhu testování, jako například rychlost lokalizace při různém počtu částic nebo vliv zvýšení nepřesnosti relativních senzorů polohy na akumulaci chyby posunu. Cílem bakalářské práce bylo zhotovení simulačního programu a následné implementaci vybraného lokalizačního algoritmu. V budoucím pokračováni této práce bude přenesení lokalizačního algoritmu na reálné zařízení.

Literatura

[1] ČAPEK, *Karel. R.U.R. : Rossum's Universal Robots, 1920.*

[2] KUDRNÁČOVÁ, Jitka. MATLAB. *Fakulta informatiky Masarykovy Univerzity* [online]. [cit. 2017-01-13]. Dostupné z: http://www.fi.muni.cz/usr/jkucera/pv109/2003p/xkudrnac_ matlab.htm

[3] FABIAN, David. *Poznámky k jazyku C++* [online]. [cit. 2017-01-13]. Dostupné z: http://kmlinux.fjfi.cvut.cz/ fabiadav/cecko/poznamky-k-jazyku-c_ plus_ plus

[4] Encoders. *INSTITUTE FOR ASTRONIMY, UNIVERSITY OF HAWAI'I* [online]. [cit. 2017-01-13]. Dostupné z: http://irtfweb.ifa.hawaii.edu/ tcs3/tcs3/0306_ conceptual_ design/Docs/05_ Encoders/encoder_ primer.pdf

[5] SKALKA, Marek. *Srovnání lokalizačních technik v robotice* [online]. [cit. 2017-01-15]. Dostupné z: http://marek.sk.sweb.cz/lokalizace/index.html

[6] HANDSCHIN, J. *Monte Carlo techniques for prediction and filtering.* Dostupné z: Auto - matica, sv. 6, p. 555–563, 1970.

[7] FOX, D., WOLFRAM, B., DELLAERT, F., THURN, S. *Monte Carlo Localization: Efficient Position Estimation for Mobile Robots.* Dostupné z: v Proceedings of the Sixteenth National Conference on Artificial Intelligence. Orlando, USA, 1999.

[8] RNDr. M.KULICH Ph.D. a Dr.rer.nat. M. SASKA. *Vybraná témata z mobilní robotiky.* Dostupné z:Seminář na VUT Brno 17. 2. 2011.

[9] KUREČKA, Aleš. *Lokalizační systém pro mobilní robotické zařízení.* Dostupné z: VŠB-TUO Ostrava, 7.5.2013. Diplomová práce.

[10] ČÁPKA, David. *IT network: 2. díl - Seznam (List) pomocí pole v C#* [online]. [cit. 2017-04-07]. Dostupné z: http://www.itnetwork.cz/csharp/kolekce-a-linq/c-sharp-tutorial-seznamy-kolekce-list

[11] KONEČNÝ, Jaromír. *Principy řízení mobilních servisních robotů.* Dostupné z: VŠB-TUO Ostrava, 22.7.2014. Disertační práce

[12] VOJÁČEK, Antonín. *Automatizace HW: Princip laserových snímačů vzdálenosti s triangulačním principem měření* [online]. 13. Červenec 2015. [cit. 2017-04-07]. Dostupné z: http://automatizace.hw.cz/mereni-a-regulace/princip-funkce-laserovych-snimacu-vzdalenosti-s-triangulacnim-principem-mereni.html

[13] *JETI model* [online]. [cit. 2017-04-07]. Dostupné z: http://www.jetimodel.com/cs/

[14] Optical encoders and LiDAR scanning. *Renishaw* [online]. [cit. 2017-04-13]. Dostupné z: http://www.renishaw.com/en/optical-encoders-and-lidar-scanning–39244

[15] LIDAR (Laser Scanner) Fundamentals. *Robots for Roboticists* [online]. [cit. 2017-04-13]. Dostupné z: http://robotsforroboticists.com/lidar-fundamentals/

[16] *Ultrasonic Ranging Module HC-SR04* [online]. [cit. 2017-04-13]. Dostupné z: http://www.micropik.com/PDF/HCSR04.pdf

[17] DUBEC, Miroslav a Jaromír SKOTNICA. *Senzory a měření: učební text* [online]. 2012. Vysoká škola báňská – Technická univerzita Ostrava [cit. 2017-04-20].

[18] *The makers workbench* [online]. [cit. 2017-04-20]. Dostupné z: http://www.themakersworkbench.com/tutorial/triggering-servo-using-hc-sr04-distance-sensor-and-arduino

A Obsah přiloženého CD

- PDF verzi bakalářské práce.
- Aplikace simulačního programu
- Zdrojové soubory simulačního programu
- Aplikace programu na výpočet π
- Zdrojové soubory programu na výpočet π

B Implementace korekce

```
public Robot[] Correction(Robot[] im, Robot main)
{
  double[] weight = new double[im.Length];
  double maxweight;
  double weight10;
  double weight30;
  double a;

  //Provede sumu cisel lidaru
  for (int i = 0; i < im.Length; i++)
  {
    a = 0;
    for (int j = 0; j < im[i].Lidar_scan.Count; j++)
    {
      a = a + ((im[i].Lidar_scan[j] - main.Lidar_scan[j]) *
          (im[i].Lidar_scan[j] - main.Lidar_scan[j]));
    }
    weight[i] = a;
  }
  //Zjisti maximum
  maxweight = weight[0];
  for (int g = 0; g < im.Length; g++)
  {
    if (maxweight < weight[g])
    {
      maxweight = weight[g];

    }
  }
  weight10 = maxweight * 0.1;
  weight30 = maxweight * 0.3;

  //normalizuje vahu
  for (int n = 0; n < im.Length; n++)
  {
    if (weight[n] < weight10)
    {
      im[n].Weight = (float)(1.9 - (weight[n] / weight10));
    }
    if (weight[n] > weight10 && weight[n] < weight30)
    {
      im[n].Weight = (float)(1.1 - (weight[n] / weight30));
    }
    if (weight[n] > weight30)
    {
      im[n].Weight = 0;
```

```
    }
    if (im[n].Position.X < 20 || im[n].Position.X > 620 ||
      im[n].Position.Y < 20 || im[n].Position.Y > 460)
    {
      im[n].Weight = 0;
    }


  }

  return im;
}
```

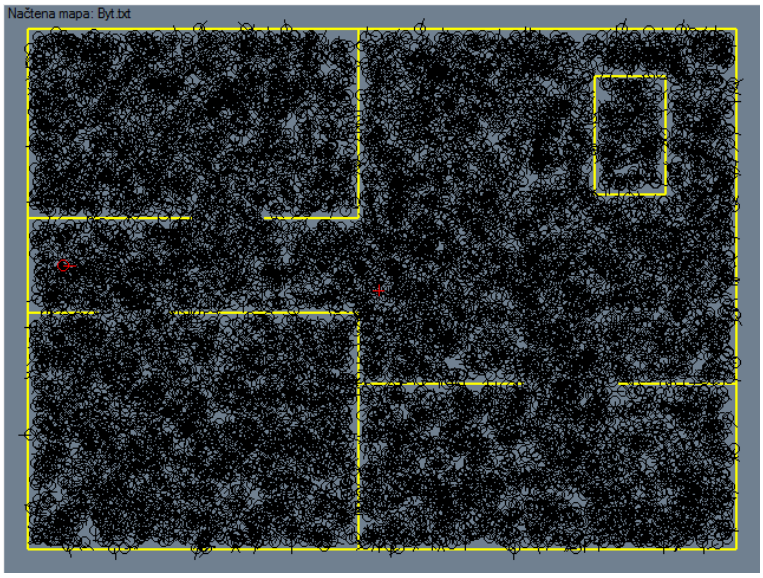
Výpis 5: Metoda pro korekci.

C Implementace převzorkování

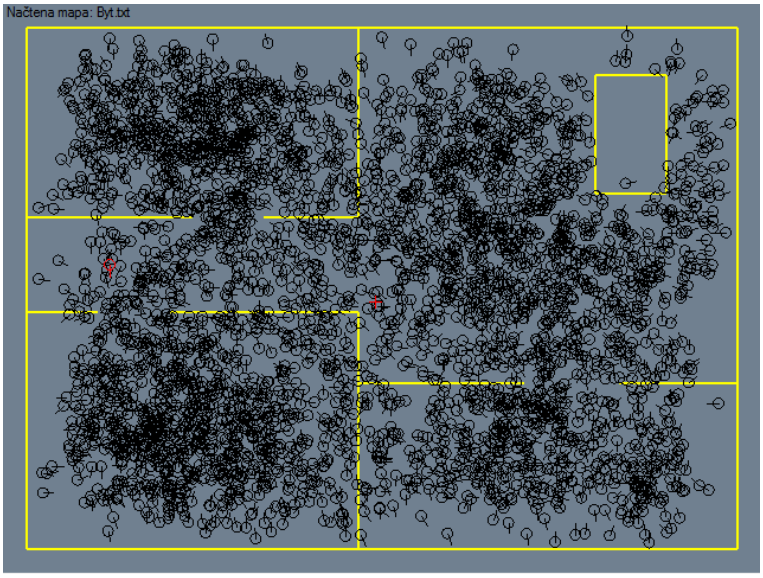
```
public Robot[] Resample(Random r, Robot[] particle)
  {
    Robot[] novevzorky = new Robot[particle.Length];
    double Sum_w = 0;

    for (int s = 0; s < particle.Length; s++)
    {
      Sum_w = Sum_w + particle[s].Weight;
    }
    double R = 0;
    double U = Sum_w / particle.Length;

    for (int x = 0; x < particle.Length; x++)
    {
      R = U * x;
      double a = 0;
      double b = 0;
      for (int e = 0; e < particle.Length; e++)
      {
        b = b + particle[e].Weight;

        if (R >= a && R < b)
        {
          novevzorky[x] = particle[e].DeepCopy();
          break;
        }
        a = b;
      }
    }
    return novevzorky;
  }
```

Výpis 6: Metoda pro převzorkování.

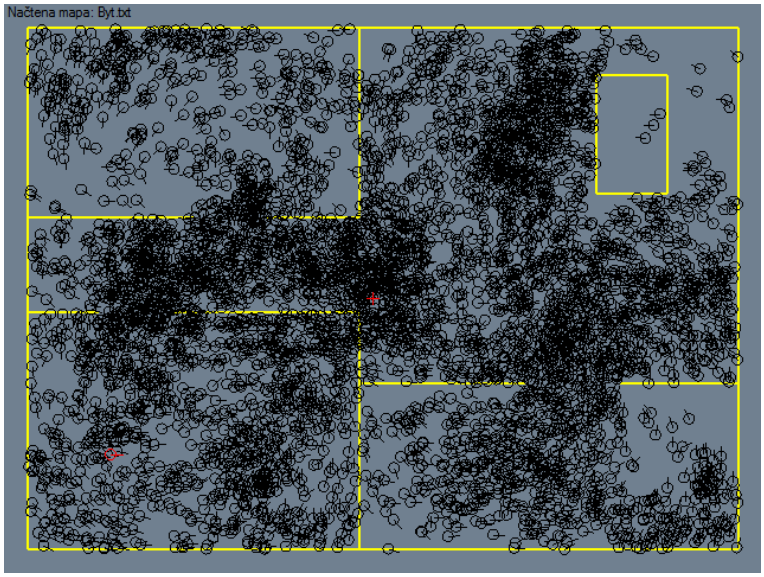
D Průbeh simulace MCL



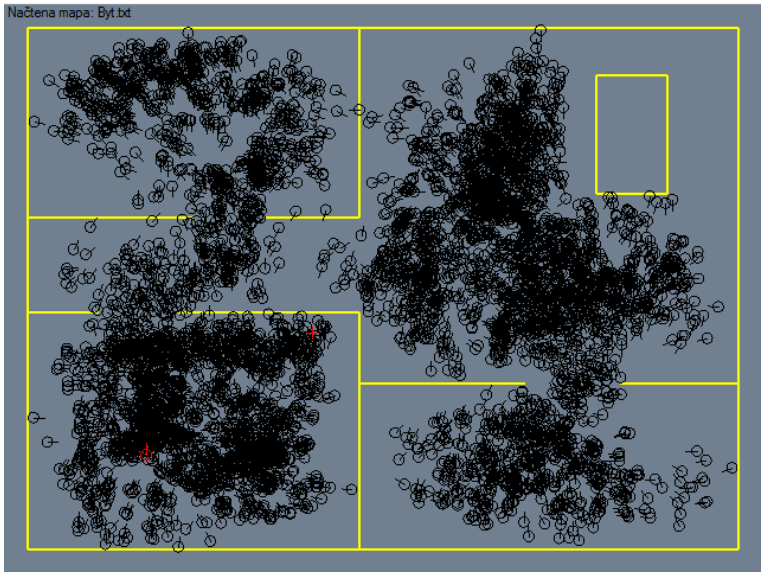
Obrázek 31: Krok 1.



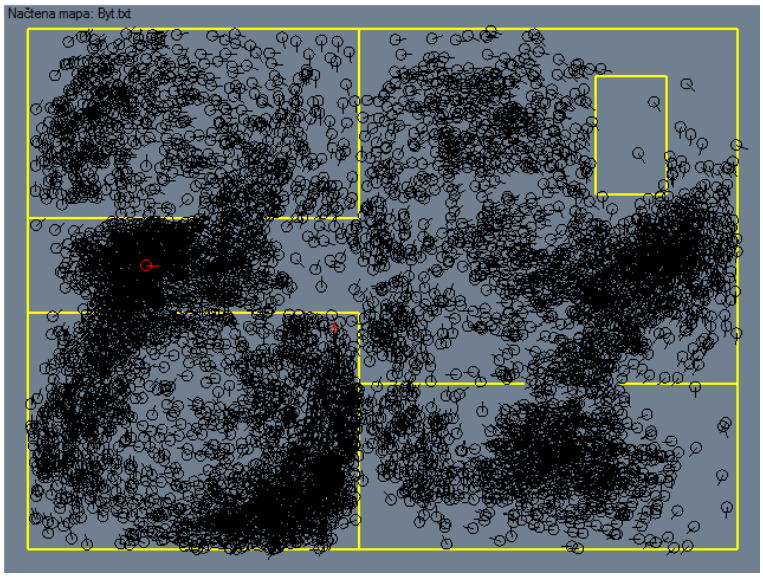
Obrázek 32: Krok 2.



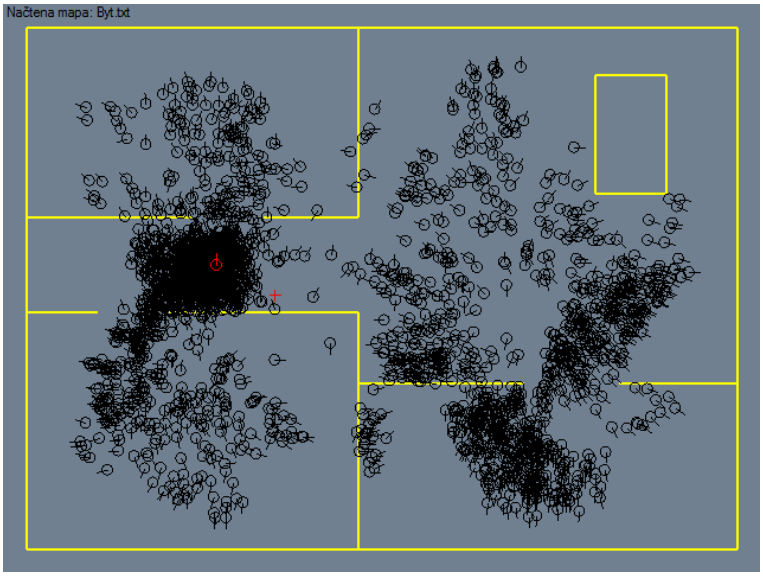
Obrázek 33: Krok 3.



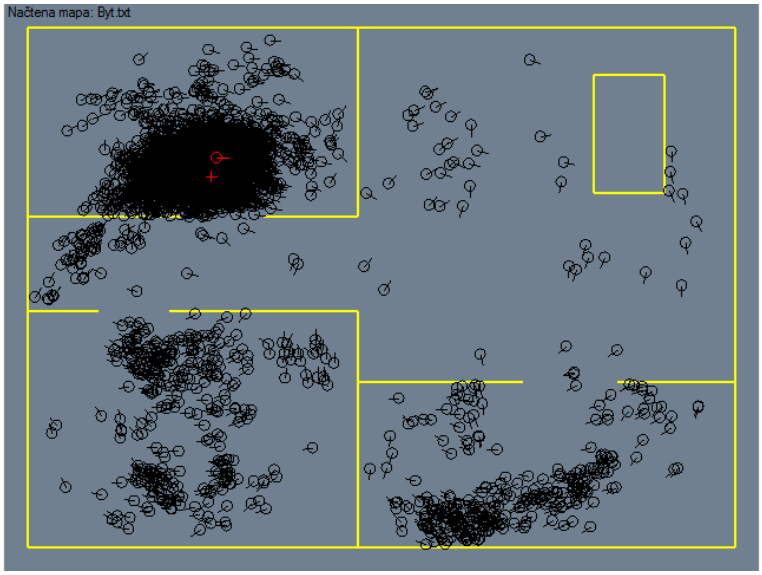
Obrázek 34: Krok 4.



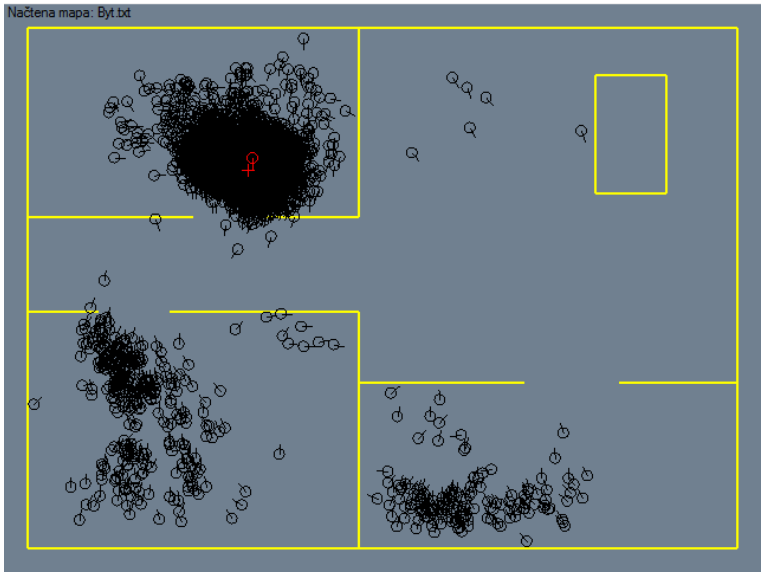
Obrázek 35: Krok 5.



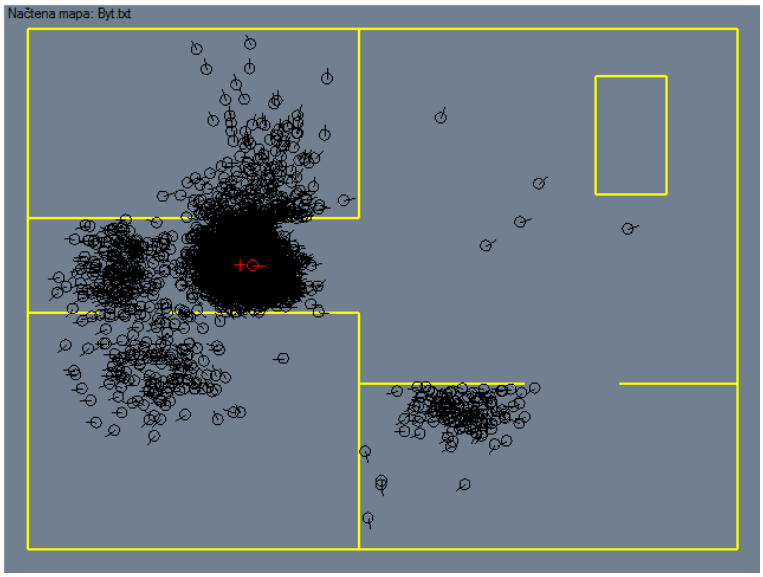
Obrázek 36: Krok 6.



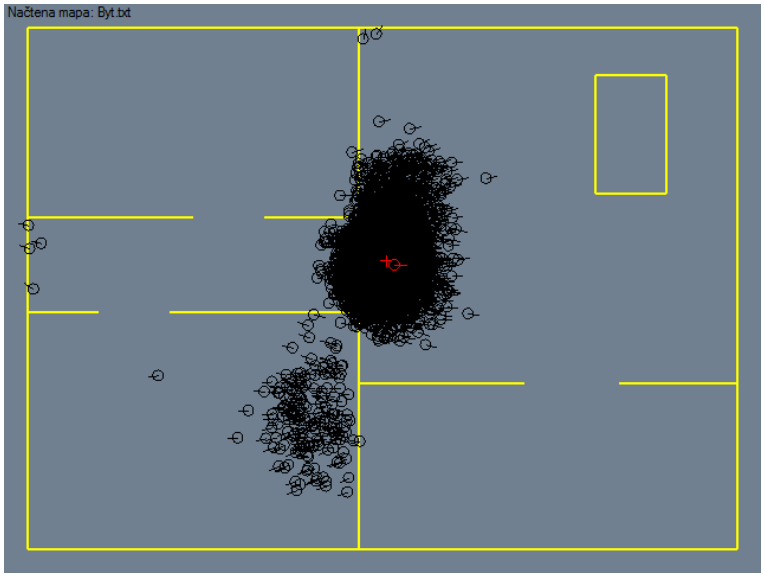
Obrázek 37: Krok 7.



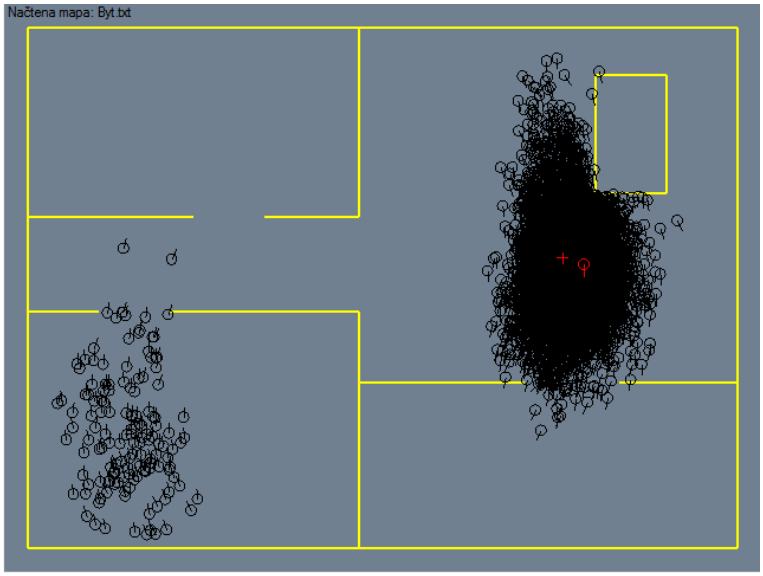
Obrázek 38: Krok 8.



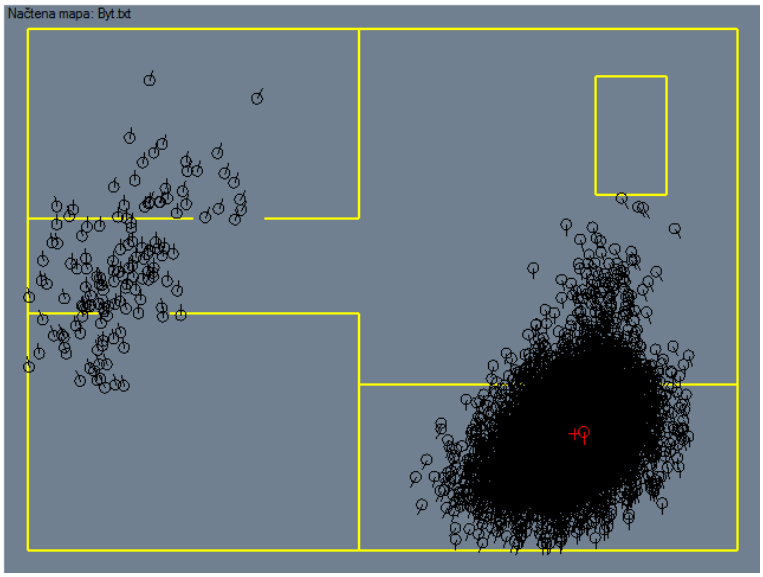
Obrázek 39: Krok 9.



Obrázek 40: Krok 10.



Obrázek 41: Krok 11.



Obrázek 42: Krok 12.